



# Введение



# Date

Класс `Date` хранит дату в миллисекундах, которые прошли с 1 января 1970 года («Unix-время»).

## Инициализация

```
Date date = new Date();
```

`date` содержит текущее время и дату

```
Date date = new Date(123456789000L);
```

В миллисекундах в `date` содержится дата  
Fri Nov 30 00:33:09 MSK 1973

```
Date date = new Date(  
    year: 12, month: 2, date: 20,  
    hrs: 10, min: 11, sec: 12);
```

В `date` содержится дата  
Wed Mar 20 10:11:12 MSK 1912

Params:

- `year` – the year minus 1900.
- `month` – the month between 0-11.
- `date` – the day of the month between 1-31.

# Конвертация в строку

Метод `toString()` преобразует дату из миллисекунд к стандартному пользовательскому формату.

Код

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

Результат

```
Mon Jan 13 16:30:58 MSK 2025
```

# Сравнение дат

Метод `getTime()` вернет количество миллисекунд, прошедших с полуночи 1 января 1970 года.

```
Date date1 = new Date();  
Date date2 = new Date();  
System.out.println(  
    (date1.getTime() > date2.getTime())?  
    "date1 позже date2" : "date1 раньше date2");
```

# Сравнение дат

`boolean equals(Object obj)`

- Проверяет даты на равенство

`boolean before(Date when)`

- Проверяет, что дата, для которой вызывается метод, раньше даты, указанной в параметре `when`

`boolean after(Date when)`

- Проверяет, что дата, для которой вызывается метод, позже даты, указанной в параметре `when`

```
Date date1 = new Date();
Date date2 = new Date();
if (date1.before(date2)) {System.out.println("date1 раньше");}
if (date1.after(date2)) {System.out.println("date1 позже");}
if (date1.equals(date2)) {System.out.println("Даты равны");}
```

# Проблемы класса Date

- Многие методы помечены аннотацией `@Deprecated`

Программный элемент, аннотированный `@Deprecated`, является тем, что программистам не рекомендуется использовать, как правило, потому, что это опасно, или потому, что существует лучшая альтернатива.

The screenshot shows an IDE interface with three main panels. The left panel, titled 'Structure', displays a list of methods for the `Date` class. The `getYear(): int` method is highlighted with a yellow box. A yellow arrow points from this box to the right panel. The middle panel shows line numbers 648, 649, 650, and 651. Line 649 contains the annotations `@Contract(pure = true)` and `@Deprecated`, both highlighted with yellow boxes. A yellow arrow points from the `@Deprecated` box to the right panel. The right panel displays the JavaDoc for the `getYear()` method, which states that it is deprecated as of JDK version 1.1 and replaced by `Calendar.get(Calendar.YEAR) - 1900`. The JavaDoc also includes the return value and a 'See Also' link to `Calendar`.

```
Structure
└─ Date
   ├── Date()
   ├── Date(long)
   ├── Date(int, int, int)
   ├── Date(int, int, int, int, int)
   ├── Date(int, int, int, int, int, int)
   ├── Date(String)
   ├── clone(): Object ↑Object
   ├── UTC(int, int, int, int, int, int): long
   ├── parse(String): long
   └─ getYear(): int
      ├── setYear(int): void
      ├── getMonth(): int
      └─ setMonth(int): void
```

```
648
649 @Contract(pure = true)
650 @Deprecated
651 public int getYear() {
    return normalize().getYear() - 1900;
}
```

Returns a value that is the result of subtracting 1900 from the year that contains or begins with the instant in time represented by this `Date` object, as interpreted in the local time zone.

Deprecated As of JDK version 1.1, replaced by `Calendar.get(Calendar.YEAR) - 1900`.

Returns: the year represented by this date, minus 1900.

See Also: `Calendar`

# Проблемы класса Date

- **Date** не поддерживает временные зоны

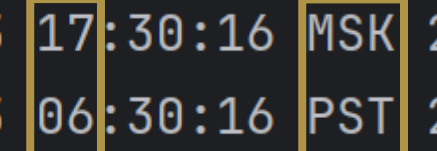
Вся работа с **Date** проходит во временной зоне, установленной в вашей системе.

Код

```
Date dataCurrent = new Date();  
System.out.println("Old - " + dataCurrent);  
  
TimeZone.setDefault(TimeZone.getTimeZone(ID: "America/Los_Angeles"));  
Date dataNewZone = new Date();  
System.out.println("New - " + dataNewZone);
```

Результат

Old - Mon Jan 13	17:30:16	MSK	2025
New - Mon Jan 13	06:30:16	PST	2025



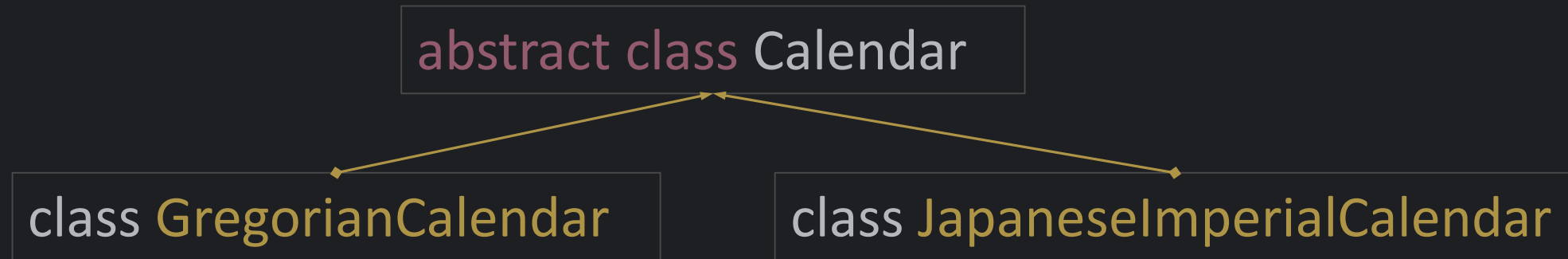


# Calendar

Абстрактный класс `Calendar` упростил работу с датами в Java.

`Calendar` имеет несколько реализаций. Далее речь пойдет только про класс `GregorianCalendar`.

`GregorianCalendar` реализует Григорианский календарь, по которому живет большинство стран мира.



Преимущества `Calendar` над `Date`.

- работает с датами в более удобном формате.
- позволяет увеличивать/уменьшать текущую дату на заданное количество дней/месяцев/лет
- позволяет проверить, является ли год високосным
- позволяет получить отдельные компоненты даты (например, получить из целой даты номер месяца)
- предоставляет систему констант для манипуляций с датами

# Инициализация

```
Calendar calendar1 = Calendar.getInstance();
```

```
Calendar calendar2 = new GregorianCalendar();
```

```
Calendar calendar3 = new GregorianCalendar(  
    year: 2024, month: 2, dayOfMonth: 12,  
    hourOfDay: 10, minute: 11, second: 12);
```



Месяца считаются с «0».

# Содержимое Calendar

Код

```
Calendar calendar1 = Calendar.getInstance();  
System.out.println(calendar1.toString());
```

Результат

```
java.util.GregorianCalendar[time=1736927326845,areFieldsSet=true,areAllFieldsSet=true,lenient=true,zone=sun.util.calendar.ZoneInfo[id="Europe/Moscow",offset=10800000,dstSavings=0,useDaylight=false,transitions=79,lastRule=null],firstDayOfWeek=2,minimalDaysInFirstWeek=4,ERA=1,YEAR=2025,MONTH=0,WEEK_OF_YEAR=3,WEEK_OF_MONTH=3,DAY_OF_MONTH=15,DAY_OF_YEAR=15,DAY_OF_WEEK=4,DAY_OF_WEEK_IN_MONTH=3,AM_PM=0,HOUR=10,HOUR_OF_DAY=10,MINUTE=48,SECOND=46,MILLISECOND=845,ZONE_OFFSET=10800000,DST_OFFSET=0]
```

# Вывод даты

Метод `getTime()` возвращает объект типа `Date`.

Код

```
Calendar calendar1 = Calendar.getInstance();
Calendar calendar2 = new GregorianCalendar();
Calendar calendar3 = new GregorianCalendar(
    year: 2024, month: 2, dayOfMonth: 12,
    hourOfDay: 10, minute: 11, second: 12);

Date date = calendar1.getTime();
System.out.println(date);

System.out.println(calendar2.getTime());
System.out.println(calendar3.getTime());
```

Результат

```
Wed Jan 15 11:05:06 MSK 2025
Wed Jan 15 11:05:06 MSK 2025
Tue Mar 12 10:11:12 MSK 2024
```

# Константы

```
Calendar calendar = new GregorianCalendar(  
    year: 2024, month: 2, dayOfMonth: 12);
```



```
Calendar calendar = new GregorianCalendar(  
    year: 2024, Calendar.MARCH, dayOfMonth: 12);
```

# Константы

## Месяца

Calendar.JANUARY  
Calendar.FEBRUARY  
Calendar.MARCH  
Calendar.APRIL  
Calendar.MAY  
Calendar.JUNE  
Calendar.JULY  
Calendar.AUGUST  
Calendar.SEPTEMBER  
Calendar.OCTOBER  
Calendar.NOVEMBER  
Calendar.DECEMBER

## Дни недели

Calendar.MONDAY  
Calendar.TUESDAY  
Calendar.WEDNESDAY  
Calendar.THURSDAY  
Calendar.FRIDAY  
Calendar.SATURDAY  
Calendar.SUNDAY

## Компоненты даты

Calendar.ERA  
Calendar.YEAR  
Calendar.MONTH  
Calendar.WEEK\_OF\_YEAR  
Calendar.WEEK\_OF\_MONTH  
Calendar.DATE  
Calendar.DAY\_OF\_MONTH  
Calendar.DAY\_OF\_YEAR  
Calendar.DAY\_OF\_WEEK  
Calendar.DAY\_OF\_WEEK\_IN\_MONTH  
Calendar.AM\_PM  
Calendar.HOUR  
Calendar.HOUR\_OF\_DAY  
Calendar.MINUTE  
Calendar.SECOND  
Calendar.MILLISECOND  
Calendar.ZONE\_OFFSET

# SET'еры и GET'еры

Код

```
Calendar calendar = new GregorianCalendar(  
    year: 2017, Calendar.JANUARY , dayOfMonth: 25);  
System.out.println(calendar.getTime());  
  
calendar.set(Calendar.HOUR, 19);  
calendar.set(Calendar.MINUTE, 42);  
calendar.set(Calendar.SECOND, 12);  
System.out.println(calendar.getTime());
```

Результат

```
Wed Jan 25 00:00:00 MSK 2017  
Wed Jan 25 19:42:12 MSK 2017
```

# SET'еры и GET'еры

Код

```
Calendar calendar = new GregorianCalendar(  
    year: 2017, Calendar.MARCH , dayOfMonth: 25);  
System.out.println(calendar.getTime());  
  
System.out.println("Год - " + calendar.get(Calendar.YEAR));  
System.out.println("Месяц - " + calendar.get(Calendar.MONTH));  
System.out.println("День - " + calendar.get(Calendar.DAY_OF_MONTH));
```

Результат

```
Sat Mar 25 00:00:00 MSK 2017  
Год - 2017  
Месяц - 2  
День - 25
```



# Изменение даты

Метод `add()` выполняет изменение даты как в большую сторону, так и в меньшую.

`add(<Изменяемое поле>, <Величина изменения>)`

Код

```
Calendar calendar = new GregorianCalendar(  
    year: 2017, Calendar.MARCH, dayOfMonth: 25);  
System.out.println(calendar.getTime());  
  
calendar.add(Calendar.DATE, amount: 10);  
System.out.println(calendar.getTime());  
  
calendar.add(Calendar.MONTH, amount: -2);  
System.out.println(calendar.getTime());
```

Результат

```
Sat Mar 25 00:00:00 MSK 2017  
Tue Apr 04 00:00:00 MSK 2017  
Sat Feb 04 00:00:00 MSK 2017
```

# Изменение даты

Метод `roll()` выполняет изменение компонента даты, не затрагивая других компонентов.

`roll(<Изменяемое поле>, <Величина изменения>)`

## Код

```
Calendar original = new GregorianCalendar(  
    year: 2017, Calendar.MARCH, dayOfMonth: 25);  
Calendar forAdd = (Calendar) original.clone();  
Calendar forRoll = (Calendar) original.clone();  
  
forAdd.add(Calendar.DATE, amount: 10);  
forRoll.roll(Calendar.DATE, amount: 10);  
  
System.out.println("Original: " + original.getTime());  
System.out.println("Add: " + forAdd.getTime());  
System.out.println("Roll: " + forRoll.getTime());
```

## Результат

```
Original: Sat Mar 25 00:00:00 MSK 2017  
Add: Tue Apr 04 00:00:00 MSK 2017  
Roll: Sat Mar 04 00:00:00 MSK 2017
```

# Форматирование даты

Класс `SimpleDateFormat` использует маску написания даты для форматирования даты при выводе и парсинга даты при вводе.

Код

```
Date date = new Date(1212121212121L);

SimpleDateFormat formater =
    new SimpleDateFormat(pattern: "HH:mm:ss dd.MM.yyyy");

System.out.println("standard: " + date);
System.out.println("format: " + formater.format(date));
```

Результат

```
standard: Fri May 30 08:20:12 MSK 2008
format: 08:20:12 30.05.2008
```

# Форматирование даты

## Шаблон

dd-MM-yyyy

yyyy MMMM dd

HH:mm:ss.SSS

yyyy-MM-dd HH:mm:ss

yyyy-MM-dd HH:mm:ss.SSS

yyyy-MM-dd HH:mm:ss.SSS Z

## Пример

01-11-2020

2008 мая 30

23:59:59.999

2018-11-30 03:09:02

2016-03-01 01:20:47.999

2013-13-13 23:59:59.999 +0100

# Парсинг строки

Код

```
9 ▶ public static void main(String[] args) {  
10  
11     String str = "Sat, April 4, 2020";  
12     SimpleDateFormat formatter =  
13         new SimpleDateFormat(  
14             pattern: "EEE, MMMM d, yyyy", Locale.ENGLISH),  
15  
16     try {  
17         Date date = formatter.parse(str);  
18         System.out.println(date);  
19     }  
20     catch (ParseException e) {  
21         e.printStackTrace();  
22     }  
23 }
```

Задает  
языковой  
стандарт,  
вместо  
стандарта по  
умолчанию

Результат

Sat Apr 04 00:00:00 MSK 2020

# Парсинг строки

Код

```
9 ▶ public static void main(String[] args) {
10
11     String str = "Sat, April 4, 2020";
12     SimpleDateFormat formatter =
13         new SimpleDateFormat(
14             pattern: "EEE, MMMM d, yyyy");//, Locale.ENGLISH);
15
16     try {
17         Date date = formatter.parse(str);
18         System.out.println(date);
19     }
20     catch (ParseException e) {
21         e.printStackTrace();
22     }
23 }
```

Убрали явное  
указание  
языкового  
стандарта

Результат

```
java.text.ParseException Create breakpoint: Unparseable date: "Sat, April 4, 2020"
    at java.base/java.text.DateFormat.parse(DateFormat.java:403)
    at org.example.Main.main(Main.java:17)
```

# SimpleDateFormat и Calendar

Код

```
Calendar calendar =  
    new GregorianCalendar(year: 216, Calendar.AUGUST, dayOfMonth: 2);  
calendar.set(Calendar.ERA, GregorianCalendar.BC);  
  
DateFormat format =  
    new SimpleDateFormat(pattern: "dd MMM yyyy GG");  
System.out.println(format.format(calendar.getTime()));
```

Результат

```
02 авг. 216 до н. э.
```

# TimeZone

Абстрактный класс `TimeZone` предназначен для совместного использования с классами `Calendar` и `DateFormat`. Статический метод `getDefault()` возвращает экземпляр наследника `TimeZone` с настройками, взятыми из операционной системы, под управлением которой работает JVM.

Код

```
final int HOUR = 60*60*1000;

TimeZone timeZone = TimeZone.getDefault();
System.out.println(timeZone.getID());
System.out.println(timeZone.getDisplayName());
System.out.println(timeZone.getRawOffset()/HOUR);
```

Результат

```
Europe/Moscow
Москва, стандартное время
3
```



# TimeZone

Статический метод `getTimeZone()` возвращает экземпляр наследника `TimeZone` с настройками указанной временной зоны.

Код

```
final int HOUR = 60*60*1000;

TimeZone timeZone = TimeZone.getTimeZone(ID: "Europe/Berlin");
System.out.println(timeZone.getID());
System.out.println(timeZone.getDisplayName());
System.out.println(timeZone.getRawOffset()/HOUR);
```

Результат

```
Europe/Berlin
Центральная Европа, стандартное время
1
```

# TimeZone

Набор ID временных зон для `getTimeZone()`, в документации явно не описывается. Его можно получить с помощью статического метод `getAvailableIds()`.

`String[] getAvailableIds()` - Возвращает полный список временных зон

`String[] getAvailableIds(int rawOffset)` - Возвращает список временных зон с указанным смещением `rawOffset`

## Код

```
final int HOUR = 60*60*1000;

String[] zones =
    TimeZone.getAvailableIds(rawOffset: 3*HOUR);
for (String zone : zones) {
    System.out.println(zone);
}
```

## Результат

Africa/Addis_Ababa	Asia/Kuwait
Africa/Asmara	Asia/Qatar
Africa/Asmera	Asia/Riyadh
Africa/Dar_es_Salaam	EAT
Africa/Djibouti	Etc/GMT-3
Africa/Kampala	Europe/Istanbul
Africa/Mogadishu	Europe/Kirov
Africa/Nairobi	Europe/Minsk
Antarctica/Syowa	Europe/Moscow
Asia/Aden	Europe/Simferopol
Asia/Amman	Europe/Volgograd
Asia/Baghdad	Indian/Antananarivo
Asia/Bahrain	Indian/Comoro
Asia/Damascus	Indian/Mayotte
Asia/Istanbul	Turkey
	W-SU

# TimeZone и Calendar. Смена часового пояса

Код	Результат
-----	-----------

13	<code>Calendar time = new GregorianCalendar();</code>
14	<code>//Задаем московское время</code>
15	<code>time.setTimeZone(TimeZone.getTimeZone(ID: "Europe/Moscow"));</code>
16	<code>time.set(Calendar.HOUR_OF_DAY, 11);</code>
17	<code>time.set(Calendar.MINUTE, 0);</code>
18	<code>time.set(Calendar.SECOND, 0);</code>
19	<code>//Выводим</code>
20	<code>System.out.println("MSK");</code>
21	<code>System.out.println(time.getTime());</code> //дата + время
22	<code>System.out.println(time.getTimeInMillis());</code> //миллисекунды
23	<code>System.out.println(time.getTimeZone().getID());</code> //часовой пояс
24	<code>//Меняем часовой пояс</code>
25	<code>time.setTimeZone(TimeZone.getTimeZone(ID: "Europe/Berlin"));</code>
26	<code>//Выводим</code>
27	<code>System.out.println("BER");</code>
28	<code>System.out.println(time.getTime());</code>
29	<code>System.out.println(time.getTimeInMillis());</code>
30	<code>System.out.println(time.getTimeZone().getID());</code>

MSK
Mon Jan 20 11:00:00 MSK 2025
1737360000895
Europe/Moscow
BER
Mon Jan 20 11:00:00 MSK 2025
1737360000895
Europe/Berlin

# TimeZone и Calendar. Смена часового пояса

Код

```
36  Calendar time = new GregorianCalendar();
37  //Задаем московское время
38  time.setTimeZone(TimeZone.getTimeZone(ID: "Europe/Moscow"));
39  time.set(Calendar.HOUR_OF_DAY, 11);
40  time.set(Calendar.MINUTE, 0);
41  time.set(Calendar.SECOND, 0);
42  //Выводим
43  System.out.println("MSK");
44  //System.out.println(time.getTime()); //дата + время
45  //System.out.println(time.getTimeInMillis()); //миллисекунды
46  //System.out.println(time.getTimeZone().getID()); //часовой пояс
47
48  //Меняем часовой пояс
49  time.setTimeZone(TimeZone.getTimeZone(ID: "Europe/Berlin"));
50  //Выводим
51  System.out.println("BER");
52  System.out.println(time.getTime());
53  System.out.println(time.getTimeInMillis());
54  System.out.println(time.getTimeZone().getID());
```

Результат

```
MSK
BER
Mon Jan 20 13:00:00 MSK 2025
1737367200467
Europe/Berlin
```

Комментируем

# TimeZone и Calendar. Смена часового пояса

Код

```
59  Calendar time = new GregorianCalendar();
60  //Задаем московское время
61  time.setTimeZone(TimeZone.getTimeZone(ID: "Europe/Moscow"));
62  time.set(Calendar.HOUR_OF_DAY, 11);
63  time.set(Calendar.MINUTE, 0);
64  time.set(Calendar.SECOND, 0);
65  //Выводим
66  System.out.println("MSK");
67  System.out.println(time.get(Calendar.HOUR_OF_DAY));
68
69  //Меняем часовой пояс
70  time.setTimeZone(TimeZone.getTimeZone(ID: "Europe/Berlin"));
71  //Выводим
72  System.out.println("BER");
73  System.out.println(time.getTime());
74  System.out.println(time.getTimeInMillis());
75  System.out.println(time.getTimeZone().getID());
```

Результат

```
MSK
11
BER
Mon Jan 20 11:00:00 MSK 2025
1737360000771
Europe/Berlin
```

Выведем часы

# TimeZone и Calendar. Смена часового пояса

## Код

```
80  Calendar time = new GregorianCalendar();
81  //Задаем московское время
82  time.setTimeZone(TimeZone.getTimeZone( ID: "Europe/Moscow"));
83  time.set(Calendar.HOUR_OF_DAY, 11);
84  time.set(Calendar.MINUTE, 0);
85  time.set(Calendar.SECOND, 0);
86  //Выводим
87  System.out.println("MSK");
88  System.out.println(time.get(Calendar.HOUR_OF_DAY));
89  System.out.println(time.getTimeInMillis());
90
91  //Меняем часовой пояс
92  time.setTimeZone(TimeZone.getTimeZone( ID: "Europe/Berlin"));
93  //Выводим
94  System.out.println("BER");
95  System.out.println(time.getTime());
96  System.out.println(time.getTimeInMillis());
97  System.out.println(time.getTimeZone().getID());
98  //Поменяем время
99  time.set(Calendar.HOUR_OF_DAY, 11);
100 System.out.println("BER v2");
101 System.out.println(time.getTime());
102 System.out.println(time.getTimeInMillis());
103 System.out.println(time.getTimeZone().getID());
```

## Результат

```
MSK
11
1737446400131
BER
Tue Jan 21 11:00:00 MSK 2025
1737446400131
Europe/Berlin
BER v2
Tue Jan 21 13:00:00 MSK 2025
1737453600131
Europe/Berlin
```

Чудо!

Внесем изменение в  
компонент времени

# TimeZone и Calendar. Вывод времени в разных часовых поясах

Класс `SimpleDateFormat` может конвертировать время в зависимости от часового пояса. Метод `setTimeZone(TimeZone zone)` устанавливает часовой пояс для объекта `SimpleDateFormat`.

## Код

```
109 //Задаем московское время
110 Calendar time = new GregorianCalendar(
111     year: 2025, Calendar.JANUARY, dayOfMonth: 20,
112     hourOfDay: 11, minute: 0, second: 0);
113 //Задаем формат времени
114 TimeZone myTZ = time.getTimeZone();
115 DateFormat format = new SimpleDateFormat(
116     pattern: "dd.MM.yyyy HH:mm:ss Z");
117 format.setTimeZone(myTZ);
118 //Выводим
119 System.out.println("My TimeZone");
120 System.out.println(format.format(time.getTime()));
121 //Меняем часовой пояс
122 TimeZone berlinTZ =
123     TimeZone.getTimeZone(ID: "Europe/Berlin");
124 format.setTimeZone(berlinTZ);
125 //Выводим
126 System.out.println("BER");
127 System.out.println(format.format(time.getTime()));
```

## Результат

```
My TimeZone
20.01.2025 11:00:00 +0300
BER
20.01.2025 09:00:00 +0100
```

# TimeZone и Calendar. Работа со строковой формой.

Представим, что время и дата хранятся в строковом представлении.

Код

```
135 String berlinTimeStr = "20.01.2025 11:00:00 +0100";
136
137 DateFormat format = new SimpleDateFormat(
138     pattern: "dd.MM.yyyy HH:mm:ss Z");
139 Date time = format.parse(berlinTimeStr);
140
141 TimeZone berlinTZ =
142     TimeZone.getTimeZone(ID: "Europe/Berlin");
143 TimeZone moscowTZ =
144     TimeZone.getTimeZone(ID: "Europe/Moscow");
145
146 format.setTimeZone(berlinTZ);
147 System.out.println("Berlin: " + format.format(time));
148 format.setTimeZone(moscowTZ);
149 System.out.println("Moscow: " + format.format(time));
```

Результат

```
Berlin: 20.01.2025 11:00:00 +0100
Moscow: 20.01.2025 13:00:00 +0300
```

ВАЖНО!  
Часовой пояс указан явно



## TimeZone и Calendar. Работа со строковой формой.

Код

```
135 String berlinTimeStr = "20.01.2025 11:00:00";
136
137 DateFormat format = new SimpleDateFormat(
138     pattern: "dd.MM.yyyy HH:mm:ss");
139 Date time = format.parse(berlinTimeStr);
140
141 TimeZone berlinTZ =
142     TimeZone.getTimeZone(ID: "Europe/Berlin");
143 TimeZone moscowTZ =
144     TimeZone.getTimeZone(ID: "Europe/Moscow");
145
146 format.setTimeZone(berlinTZ);
147 System.out.println("Berlin: " + format.format(time));
148 format.setTimeZone(moscowTZ);
149 System.out.println("Moscow: " + format.format(time));
```

Результат

```
Berlin: 20.01.2025 09:00:00
Moscow: 20.01.2025 11:00:00
```

Нет часового пояса



# Чем плох Calendar?

## 1. Сложность и неинтуитивность API

Методы неочевидны.

Требуется запоминание множества констант (например, `Calendar.MONTH`, `Calendar.DAY_OF_MONTH`).

Нумерация месяцев начинается с 0, а дней - с 1.

## 2. Мутабельность (изменяемость)

## 3. Проблемы с потокобезопасностью

## 4. Устаревший дизайн

`Calendar` был введен в Java 1.1 как замена для `java.util.Date`, но устарел к появлению Java 8.

## 5. Отсутствие поддержки временных зон

Конвертация между временными зонами требует дополнительных усилий.

## 6. Неудобство для работы с датами и временем

`Calendar` объединяет дату и время в одном объекте, что не всегда удобно.

## 7. Проблемы с форматированием

Для форматирования дат и времени в `Calendar` требуется использование класса `SimpleDateFormat`.

## 8. Низкая производительность

## 9. Отсутствие поддержки современных стандартов

`Calendar` не поддерживает современные стандарты, такие как ISO-8601.

## 10. Проблемы совместимости с `java.util.Date`

# java.Time

Спустя 15 лет после выхода `Calendar` был представлен `Java Date Time API`: набор классов, которые должны решить все возможные проблемы со временем.

`Java Date Time API` состоит из нескольких пакетов.

Пакет `java.time` — базовый пакет для `Java Date Time API`: в нем содержатся такие классы как `LocalDate`, `LocalTime`, `LocalDateTime`, `Instant`, `Period`, `Duration`. Все объекты этих классов — `immutable`: их нельзя изменить после создания.

Пакет `java.time.format` содержит в себе классы для форматирования времени: преобразования времени (и даты) в текстовую строку и обратно. Например, в нем содержится такой универсальный класс как `DateTimeFormatter`, который пришел на смену `SimpleDateFormat`.

Пакет `java.time.zone` содержит классы для работы с часовыми поясами (time zones). Он содержит такие классы как `TimeZone` и `ZonedDateTime`. Если вы пишете код для сервера, клиенты которого находятся в разных частях света, эти классы вам очень понадобятся.

# LocalDate. Инициализация

Класс `LocalDate` создан для работы с датой.

Объекты класса не изменяются после создания (класс `LocalDate` `immutable`).

Это обеспечивает простоту и надежность использования, а так же потокобезопасность.

Инициализация:

`LocalDate now()` - Статический метод, возвращающий текущую дату

`LocalDate of(int year, int month, int dayOfMonth)` - Статический метод, возвращающий конкретную дату

Код

```
LocalDate date1 = LocalDate.now();
LocalDate date2 = LocalDate.of(
    year: 2020, month: 1, dayOfMonth: 1);
LocalDate date3 = LocalDate.of(
    year: 2023, Month.APRIL, dayOfMonth: 23);

System.out.println(date1);
System.out.println(date2);
System.out.println(date3);
```

Результат

```
2025-01-22
2020-01-01
2023-04-23
```

# LocalDate. Получение фрагментов даты

<code>int getYear()</code>	- Возвращает год из конкретной даты
<code>Month getMonth()</code>	- Возвращает месяц даты — одну из специальных констант JANUARY, FEBRUARY, ...;
<code>int getMonthValue()</code>	- Возвращает номер месяца из даты. Январь == 1.
<code>int getDayOfMonth()</code>	- Возвращает номер дня в месяце
<code>int getDayOfYear()</code>	- Возвращает номер дня с начала года
<code>DayOfWeek getDayOfWeek()</code>	- Возвращает день недели: одну из специальных констант MONDAY, TUESDAY, ...;
<code>IsoEra getEra()</code>	- Возвращает эру: константа BC (Before Current Era) и CE(Current Era)

## LocalDate. Изменение даты

<code>LocalDate plusDays(int days)</code>	- Добавляет определенное количество дней к дате
<code>LocalDate plusWeeks(int weeks)</code>	- Добавляет недели к дате
<code>LocalDate plusMonths(int months)</code>	- Добавляет месяцы к дате
<code>LocalDate plusYears(int years)</code>	- Добавляет годы к дате
<code>LocalDate minusDays(int days)</code>	- Отнимает дни от даты
<code>LocalDate minusWeeks(int weeks)</code>	- Отнимает недели от даты
<code>LocalDate minusMonths(int months)</code>	- Отнимает месяцы от даты
<code>LocalDate minusYears(int years)</code>	- Отнимает годы от даты

# LocalTime. Инициализация

Класс `LocalTime` создан для работы со временем.

Объекты класса не изменяются после создания (класс `LocalTime immutable`).  
Это обеспечивает простоту и надежность использования, а так же потокобезопасность.

Инициализация:

`LocalTime now()`

- Статический метод, возвращающий текущее время

`LocalTime of(int hour, int minute, int second)`

- Статический метод, возвращающий конкретное время

Код

```
LocalTime time1 = LocalTime.now();
LocalTime time2 = LocalTime.of(
    hour: 11, minute: 0, second: 1);
LocalTime time3 = LocalTime.of(
    hour: 12, minute: 0, second: 0, nanoOfSecond: 100);

System.out.println(time1);
System.out.println(time2);
System.out.println(time3);
```

Результат

```
12:27:08.837800900
11:00:01
12:00:00.000000100
```

# LocalTime. Получение фрагментов времени

`int getHour()` - Возвращает часы

`int getMinute()` - Возвращает минуты

`int getSecond()` - Возвращает секунды

`int getNano()` - Возвращает наносекунды

Код

```
LocalTime now = LocalTime.now();  
System.out.println(now.getHour());  
System.out.println(now.getMinute());  
System.out.println(now.getSecond());  
System.out.println(now.getNano());
```

Результат

```
12  
34  
56  
789378500
```



# LocalTime. Изменение времени

<code>LocalTime plusHours(int hours)</code>	- Добавляет часы
<code>LocalTime plusMinutes(int minutes)</code>	- Добавляет минуты
<code>LocalTime plusSeconds(int seconds)</code>	- Добавляет секунды
<code>LocalTime plusNanos(int nanos)</code>	- Добавляет наносекунды
<code>LocalTime minusHours(int hours)</code>	- Вычитает часы
<code>LocalTime minusMinutes(int minutes)</code>	- Вычитает минуты
<code>LocalTime minusSeconds(int seconds)</code>	- Вычитает секунды
<code>LocalTime minusNanos(int nanos)</code>	- Вычитает наносекунды

## Код

```
LocalTime time = LocalTime.now();
LocalTime time2 =
    time.plusHours( hoursToAdd: 2);
LocalTime time3 =
    time.minusMinutes( minutesToSubtract: 40);
LocalTime time4 =
    time.plusSeconds( secondstoAdd: 3600);

System.out.println(time);
System.out.println(time2);
System.out.println(time3);
System.out.println(time4);
```

## Результат

```
12:46:36.215528400
14:46:36.215528400
12:06:36.215528400
13:46:36.215528400
```

# LocalDateTime. Инициализация

Класс `LocalDateTime` объединяет в себе возможности классов `LocalDate` и `LocalTime`.

Инициализация:

- |  |  |
|--|--|
| <code>LocalDateTime now()</code>   | - Статический метод, возвращающий текущие дату и время |
| <code>LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond)</code> | - Статический метод, возвращающий конкретное время     |
| <code>LocalDateTime of(LocalDate date, LocalTime time)</code>  | - Статический метод, возвращающий конкретное время     |

Код

```
LocalDateTime now = LocalDateTime.now();
LocalDateTime sometime = LocalDateTime.of(
    year: 2025, month: 1, dayOfMonth: 25,
    hour: 11, minute: 0, second: 1, nanoOfSecond: 100);

LocalDate date = LocalDate.of(year: 2025, month: 2, dayOfMonth: 25);
LocalTime time = LocalTime.of(hour: 12, minute: 30, second: 30);
LocalDateTime dateTime = LocalDateTime.of(date, time);

System.out.println(now);
System.out.println(sometime);
System.out.println(dateTime);
```

Результат

```
2025-01-24T15:54:34.087451400
2025-01-25T11:00:01.000000100
2025-02-25T12:30:30
```

# Instant

Класс `Instant` предназначен для работы со временем, нацеленным на процессы внутри компьютеров. Он оперирует секундами, миллисекундами и наносекундами.

Класс хранит в себе два поля:

- количество секунд, прошедшее с 1 января 1970 года
- количество наносекунд

Время считается в Unix-time: от начала 1970 года.

`Instant` — это упрощенная версия класса `Date`.

Код

```
Instant start = Instant.now();  
String s = stringConcatenation();  
Instant end = Instant.now();  
Duration duration = Duration.between(start, end);  
System.out.printf("String - %dmc. %n", duration.toMillis());
```

# Zoned и ZonedDateTime

Класс `ZonedId` отвечает за работу с часовыми поясами. На его базе работает класс `TimeZone`.

Класс `ZonedDateTime` работает со временем с учетом часового пояса.

## Код

```
//Время в Берлине
ZoneId berlinZone = ZoneId.of( zoneId: "Europe/Berlin");
ZonedDateTime berlinTime = ZonedDateTime.now(berlinZone);
System.out.println("Berlin: "+berlinTime);

//Конвертируем берлинское время в московское
ZoneId moscowZone = ZoneId.of( zoneId: "Europe/Moscow");
ZonedDateTime moscowTime =
    berlinTime.withZoneSameInstant(moscowZone);
System.out.println("Moscow: "+moscowTime);

ZonedDateTime moscowTime2 =
    berlinTime.withZoneSameLocal(moscowZone);
System.out.println("Moscow2: "+ moscowTime2);

LocalDateTime localTime = berlinTime.toLocalDateTime();
System.out.println("LocalTime: "+localTime);
```

## Результат

```
Berlin: 2025-01-24T14:43:01.566139500+01:00[Europe/Berlin]
Moscow: 2025-01-24T16:43:01.566139500+03:00[Europe/Moscow]
Moscow2: 2025-01-24T14:43:01.566139500+03:00[Europe/Moscow]
LocalTime: 2025-01-24T14:43:01.566139500
```

# DateTimeFormatter

Класс **DateTimeFormatter** по функционалу аналогичен устаревшему SimpleDateFormat, но заточен под современный API для работы со временем и датой.

Особенности **DateTimeFormatter**:

- работает с объектами java.time
- потокобезопасен
- неизменяем
- поддерживает современные стандарты оформления времени и даты
- оптимизирован

# DateTimeFormatter. Форматирование

Код

```
DateTimeFormatter dtf =  
    DateTimeFormatter.ofPattern("dd MMMM yyyy");  
LocalDate date = LocalDate.now();  
String str = date.format(dtf);  
String str2 = dtf.format(date);  
  
System.out.println("Base: " + date);  
System.out.println("Str1: " + str);  
System.out.println("Str2: " + str2);
```

Результат

```
Base: 2025-01-27  
Str1: 27 января 2025  
Str2: 27 января 2025
```

# DateTimeFormatter. Форматирование

Код

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

DateTimeFormatter dtf =
    DateTimeFormatter.ofPattern("dd MMMM yyyy");

DateTimeFormatter dtf_eng =
    DateTimeFormatter.ofPattern(
        pattern: "dd MMMM yyyy", Locale.ENGLISH);

System.out.println("Default: " + date.format(dtf));
System.out.println("English: " + date.format(dtf_eng));
```

Результат

```
Default: 27 января 2025
English: 27 January 2025
```

# DateTimeFormatter. Форматирование

Код

```
DateTimeFormatter dateTimeFormat =  
    DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm");  
DateTimeFormatter dateFormat =  
    DateTimeFormatter.ofPattern("dd MMM yyyy");  
DateTimeFormatter timeFormat =  
    DateTimeFormatter.ofPattern("HHч мм ss.SSS");  
  
LocalDate date = LocalDate.now();  
LocalTime time = LocalTime.now();  
LocalDateTime dateTime = LocalDateTime.now();  
  
System.out.println(dateTime.format(dateTimeFormat));  
System.out.println(date.format(dateFormat));  
System.out.println(time.format(timeFormat));
```

Результат

```
27.01.2025 14:50  
27 янв. 2025  
14ч 50м 59.459
```



# DateTimeFormatter. Парсинг

Код

```
DateTimeFormatter dtf =  
    DateTimeFormatter.ofPattern("dd.MM.yyyy");  
String str = "23.01.2025";  
LocalDate date = LocalDate.parse(str, dtf);  
System.out.println(date);
```

Результат

2025-01-23

Код

```
DateTimeFormatter dtf =  
    DateTimeFormatter.ofPattern(  
        pattern: "MMMM-dd-yyyy", Locale.ENGLISH);  
LocalDate date = LocalDate.parse(  
    text: "February-23-2019", dtf);  
System.out.println(date);
```

Результат

2019-02-23

# Сравнение дат и времени. Duration и Period

Для сопоставления дат и времени классы `LocalDate`, `LocalTime` и `LocalDateTime` имеют методы:

`boolean equals(Object obj)` - Проверяет объекты на равенство

`boolean isBefore(other)` - Возвращает `true`, если объект, для которого вызывается метод, раньше объекта `other`

`boolean isAfter(other)` - Возвращает `true`, если объект, для которого вызывается метод, позже объекта `other`

Так же есть два класса для работы с длительностью:

- Класс `Period` представляет количество времени в годах, месяцах и днях.
- Класс `Duration` служит для хранения продолжительности времени на основе секунд и наносекунд.

# Сравнение дат и времени. Duration и Period

Для сопоставления дат и времени классы `LocalDate`, `LocalTime` и `LocalDateTime` имеют методы:

`boolean equals(Object obj)` - Проверяет объекты на равенство

`boolean isBefore(other)` - Возвращает `true`, если объект, для которого вызывается метод, раньше объекта `other`

`boolean isAfter(other)` - Возвращает `true`, если объект, для которого вызывается метод, позже объекта `other`

Так же есть два класса для работы с длительностью:

- Класс `Period` представляет количество времени в годах, месяцах и днях.
- Класс `Duration` служит для хранения продолжительности времени на основе секунд и наносекунд.

# Duration

## Код

```
Duration oneDay = Duration.ofDays(1);
Duration oneHour = Duration.ofHours(1);
Duration oneMin = Duration.ofMinutes(1);
Duration tenSeconds = Duration.ofSeconds(10);
Duration twoSeconds =
    Duration.ofSeconds(
        seconds: 1, nanoAdjustment: 1_000_000_000);
Duration oneSecondFromMillis = Duration.ofMillis(1);
Duration oneSecondFromNanos =
    Duration.ofNanos(1000000000);
Duration oneSecond =
    Duration.of(amount: 1, ChronoUnit.SECONDS);

System.out.println("oneDay: " + oneDay);
System.out.println("oneHour: " + oneHour);
System.out.println("oneMin: " + oneMin);
System.out.println("tenSeconds: " + tenSeconds);
System.out.println("twoSeconds: " + twoSeconds);
System.out.println("oneSecondFromMillis: " + oneSecondFromMillis);
System.out.println("oneSecondFromNanos: " + oneSecondFromNanos);
System.out.println("oneSecond: " + oneSecond);
```

## Результат

```
oneDay: PT24H
oneHour: PT1H
oneMin: PT1M
tenSeconds: PT10S
twoSeconds: PT2S
oneSecondFromMillis: PT0.001S
oneSecondFromNanos: PT1S
oneSecond: PT1S
```

# Duration

Код

```
LocalTime time1 = LocalTime.now();  
LocalTime time2 = time1.plusMinutes(minutesToAdd: 2);  
Duration duration = Duration.between(time1, time2);  
  
System.out.println(duration.toMinutes());  
System.out.println(duration.toSeconds());  
System.out.println(duration.toMillis());
```

Результат

```
2  
120  
120000
```

# Period

Код

```
Period period5y4m3d = Period.of(  
    years: 5, months: 4, days: 3);  
Period period2d = Period.ofDays(2);  
Period period2m = Period.ofMonths(2);  
Period period14d = Period.ofWeeks(2);  
Period period2y = Period.ofYears(2);  
  
System.out.println(period5y4m3d);  
System.out.println(period2d);  
System.out.println(period2m);  
System.out.println(period14d);  
System.out.println(period2y);
```

Результат

```
P5Y4M3D  
P2D  
P2M  
P14D  
P2Y
```

# Period

Код

```
Period period5y4m3d =  
    Period.of(years: 5, months: 4, days: 3);  
  
int days = period5y4m3d.getDays();  
int months = period5y4m3d.getMonths();  
int year = period5y4m3d.getYears();  
long days2 = period5y4m3d.get(ChronoUnit.DAYS);  
  
System.out.println("Дни: " + days);  
System.out.println("Месяца: " + months);  
System.out.println("Годы: " + year);  
System.out.println("Дни: " + days2);
```

Результат

```
Дни: 3  
Месяца: 4  
Годы: 5  
Дни: 3
```

# Period

Код

```
LocalDate date1 =  
    LocalDate.of(year: 2025, month: 3, dayOfMonth: 1);  
LocalDate date2 =  
    LocalDate.of(year: 2025, month: 5, dayOfMonth: 12);  
Period dif = Period.between(date1, date2);  
  
System.out.println(dif);
```

Результат

P2M11D



# Конвертация java.Time в Calendar

Вариант 1. Прямая инициализация.

Код

```
LocalDate localDate = LocalDate.now();
Calendar calendar = new GregorianCalendar();
//Наполняем календарь
calendar.set(Calendar.YEAR, localDate.getYear());
calendar.set(Calendar.MONTH, localDate.getMonthValue() - 1);
calendar.set(Calendar.DAY_OF_MONTH, localDate.getDayOfMonth());

System.out.println("LocalDate: " + localDate);
System.out.println("Calendar: " + calendar.getTime());
```

1. Различается подход к нумерации месяцев.

Результат

```
LocalDate: 2025-01-29
Calendar: Wed Jan 29 14:30:38 MSK 2025
```

2. Calendar по умолчанию инициализирует время.

3. Работа с часовым поясом по умолчанию.

# Конвертация java.Time в Calendar

Вариант 2. Через приведение к `ZonedDateTime`. Часовой пояс по умолчанию.

Код

```
//Текущая дата
LocalDate localDate = LocalDate.now();
//Текущий часовой пояс
ZoneId myZoneId = ZoneId.systemDefault();
//Генерим ZonedDateTime на начало дня в нужном часовом поясе
ZonedDateTime zonedDateTime = localDate.atStartOfDay(myZoneId);
//Конвертируем в календарь
Calendar calendar = GregorianCalendar.from(zonedDateTime);

System.out.println("LocalDate: " + localDate);
System.out.println("Calendar: " + calendar.getTime());
```

Результат

```
LocalDate: 2025-01-29
Calendar: Wed Jan 29 00:00:00 MSK 2025
```

# Конвертация java.Time в Calendar

Вариант 2. Через приведение к `ZonedDateTime`. Явное указание часового пояса.

Код

```
//Текущая дата
LocalDate localDate = LocalDate.now();
//Рандомный часовой пояс
ZoneId myZoneId = ZoneId.of(zoneId: "Europe/Berlin");
//Генерим ZonedDateTime на начало дня в нужном часовом поясе
ZonedDateTime zonedDateTime = localDate.atStartOfDay(myZoneId);
//Конвертируем в календарь
Calendar calendar = GregorianCalendar.from(zonedDateTime);

System.out.println("LocalDate: " + localDate);
System.out.println("Calendar: " + calendar.getTime());
System.out.println("CalendarZoneId: " + calendar.getTimeZone().getID());
```

Результат

```
LocalDate: 2025-01-29
Calendar: Wed Jan 29 02:00:00 MSK 2025
CalendarZoneId: Europe/Berlin
```

# Конвертация Calendar в java.Time

Конвертация через приведение к `Instant`. Часовой пояс по умолчанию.

Код

```
//Инициализация календаря
Calendar calendar = Calendar.getInstance();
//Приведение календаря к Instant
Instant instant = calendar.toInstant();
//Определение часового пояса
ZoneId myZoneId = ZoneId.systemDefault();
//Приведение Instant к ZonedDateTime
ZonedDateTime zonedDateTime = instant.atZone(myZoneId);

System.out.println("Calendar: " + calendar.getTime());
System.out.println("ZonedDateTime: " + zonedDateTime);
```

Результат

```
Calendar: Wed Jan 29 15:22:02 MSK 2025
ZonedDateTime: 2025-01-29T15:22:02.261+03:00[Europe/Moscow]
```

# Конвертация Calendar в java.Time

Конвертация через приведение к `Instant`. С указанием часового пояса.

Код

```
//Инициализация календаря
Calendar calendar = Calendar.getInstance(
    TimeZone.getTimeZone(ID: "Europe/Berlin"));
//Приведение календаря к Instant
Instant instant = calendar.toInstant();
//Определение часового пояса
ZoneId calendarZoneId = calendar.getTimeZone().toZoneId();
//Приведение Instant к ZonedDateTime
ZonedDateTime zonedDateTime = instant.atZone(calendarZoneId);

System.out.println("Calendar: " + calendar.getTime());
System.out.println("ZonedDateTime: " + zonedDateTime);
```

Результат

```
Calendar: Wed Jan 29 15:26:42 MSK 2025
ZonedDateTime: 2025-01-29T13:26:42.219+01:00[Europe/Berlin]
```

# Задачи для самостоятельной работы

1. Определите разницу во времени между двумя часовыми поясами (например, "Europe/London" и "Asia/Tokyo").
2. Распарсите строку "11ч. 23м. 44с." в объект `LocalTime` и выведите его.
3. Создайте объект `Period`, представляющий интервал в 2 года, 3 месяца и 10 дней, и добавьте его к текущей дате.
4. Для объекта `Calendar` проверьте, является ли текущий год високосным.
5. Напишите программу, которая вычисляет возраст человека по дате рождения.
6. Создайте программу, которая выводит количество дней до ближайшего дня рождения.
7. Напишите программу, которая выводит все пятницы 13-го в указанном году.
8. Конвертируйте объект `Date` в `LocalDateTime` и наоборот.