Date / Time

Object-Oriented Programming

SoftEng
http://softeng.polito.it

Version 1.3.0 - April 2025

Time and Date APIs

- There are several APIs that introduced in different steps following each other in time:
  - Time stamps (in **`java.lang.System`**)
  - **`java.util.Date`**
  - **`java.util.Calendar`**
  - **`java.time`**

System time stamps

- **`System`** class provides two methods:

`currentTimeMillis()`

- the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC

`nanoTime()`

- current value of the running JVM's high-resolution time source, in nanosecond
- There is no absolute reference

Date

- Original date class **java.util.Date**
  - Encapsulate a `long` time-stamp
  - Unsuitable for internationalization
    - Several methods are deprecated
- May 6, 2015 would be:

Deprecated

```
Date d = new Date(115,4,6);
String s = d.toString();
```

```
"Wed May 06 00:00:00 CEST 2015"
```

`Calendar`

- Abstract class, with one concrete implementation: **`GregorianCalendar`**
- Represents a date with fields
  - **`YEAR, MONTH, DAY_OF_MONTH, HOUR`**…
- Can be manipulate
  - **`get(field)`**
  - **`set(field, value)`**
  - **`add(field, delta)`**

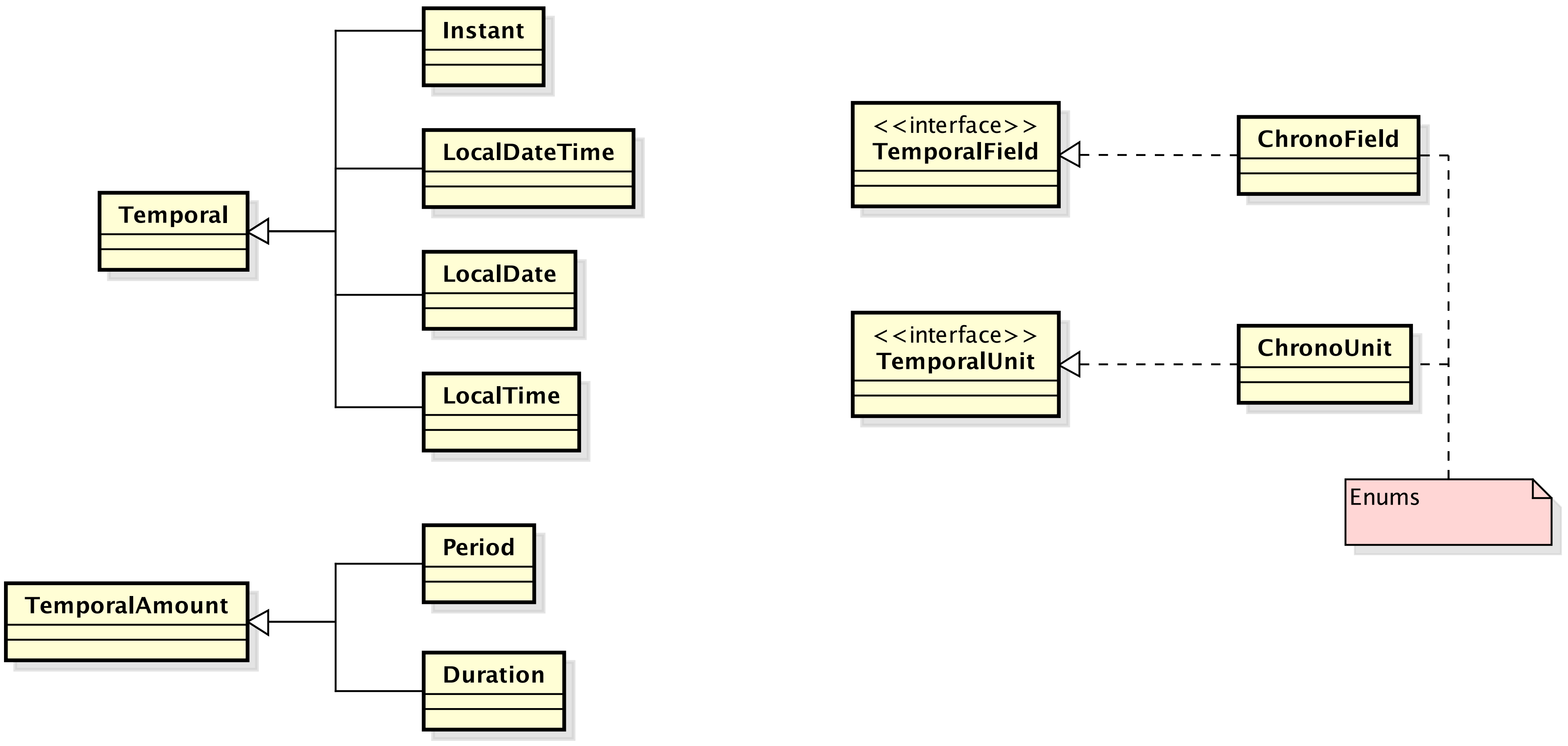
New Date and Time

- Package **`java.time`**
  - Introduced in Java 8
- Guiding principles
  - Simplicity
  - Consistency
- All classes are immutable

Main classes

- Temporal points
  - `Instant`
  - `LocalDate`
  - `LocalDateTime`
  - `LocalTime`
  - `ZonedDateTime`
- Temporal intervals
  - `Duration` (time based)
  - `Period` (date based)

Main classes



Time points factory methods

Method	Purpose
`of()`	Create instance from a set of specific parameters, with validation
`from()`	Convert from another class with possible loss of information
`parse()`	Parse a string to build an instance
`now()`	Create an instance representing the current time / date. Can accept a **ZoneId**

Comparison

Method	Purpose
`isBefore()`	Checks if this time/date is before the specified time/date
`isAfter()`	Checks if this time/date is after the specified time/date
`isEqual()`	Checks if this time/date is the same as the specified time/date
`compareTo()`	Compares to to other time/date

Changing

Method	Purpose
`minus()`	Returns a new date/time built by removing a specific amount of date/time
`plus()`	Returns a new date/time built by adding a specific amount of date/time
`with()`	Returns a new date/time modified as specified by a temporal adjuster

`plus / minus`

- Plus/Minus
  - **`long amountToSubtract,`**
  - **`TemporalUnit unit`**
    - E.g. **`ChronoUnit.DAYS`**
- Plus/Minus
  - **`TemporalAmount amount`**
    - Either a **`Duration`** or a **`Period`**

Temporal adjusters

- Factory methods in class **`TemporalAdjusters`**, e.g.
  - **`firstDayOfMonth()`**
  - **`firstDayOfNextMonth()`**
  - **`firstInMonth(DayOfWeek dayOfWeek)`**
  - **`lastDayOfMonth()`**
  - …
- Can be used as arguments to **`with()`**

DoW and Month

- Are represented by enums:
  - **`DayOfWeek`**
  - **`Month`**
- Can be converted to string
  - **`getDisplayName(style,locale)`**
  - style is one of
    - **`TextStyle.FULL`**
    - **`TextStyle.NARROW`**
    - **`TextStyle.SHORT`**

Examples

```
LocalDate today = LocalDate.now();
LocalDate tomorrow = today
             .plus(1,ChronoUnit.DAYS);
LocalDate inTwoWeeks = today
                          .plusDays(14);
LocalDate firstMon = today
   .with(TemporalAdjusters
           .firstInMonth(
                      DayOfWeek.MONDAY));
```

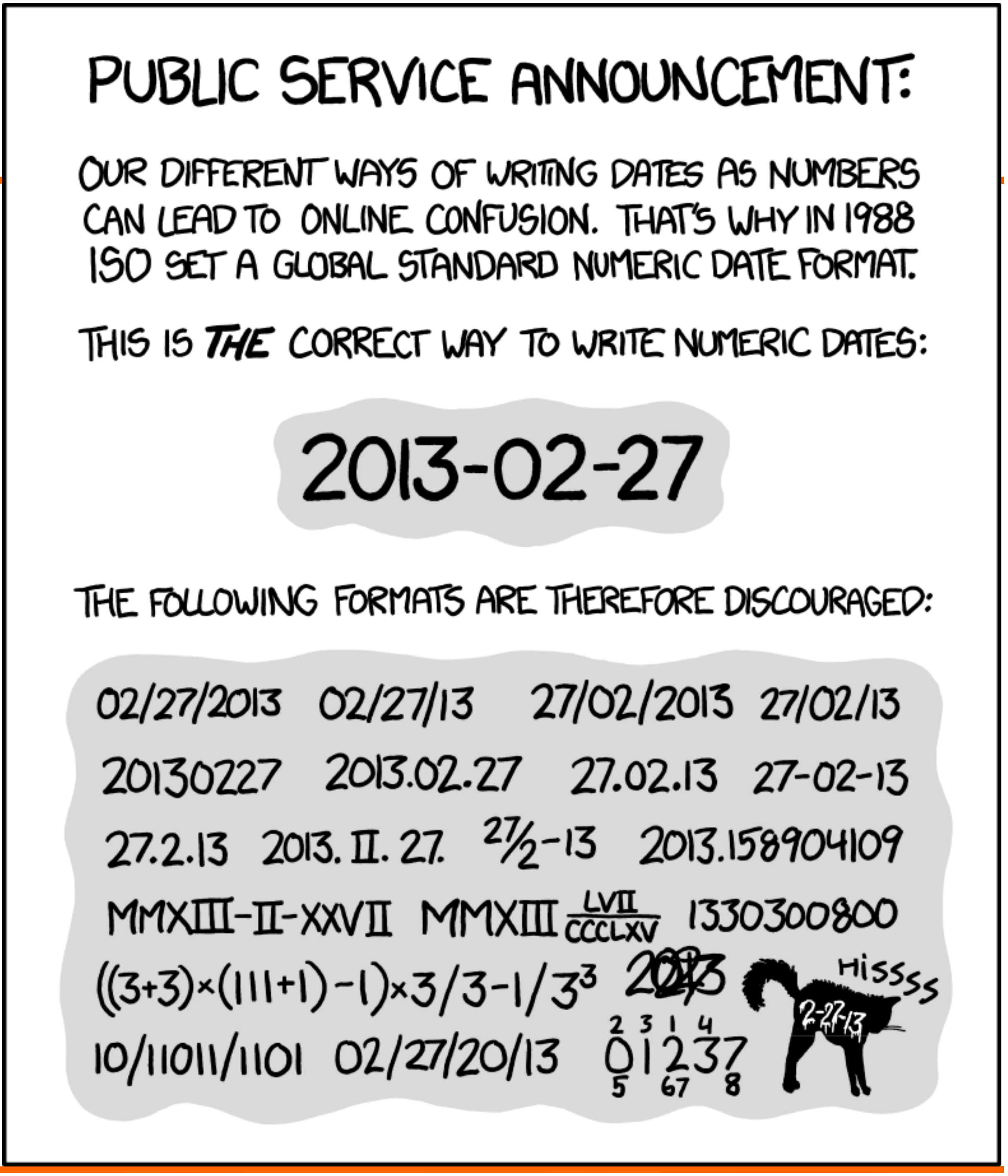
Locale

- Class `Locale` represents a specific geographical, political, or cultural region
- Used to perform *locale-sensitive* operations
  - Date formats
  - DoW, Month names
  - Decimal separators

Locale definition

- Predefined constants, e.g.,
  - **`Locale.US`**, **`Locale.ITALIAN`**
- Constructors
  - Language: 2 or 3 chars code
  - Country: 2 chars or 3 digits
  - Variant: optional additional spec

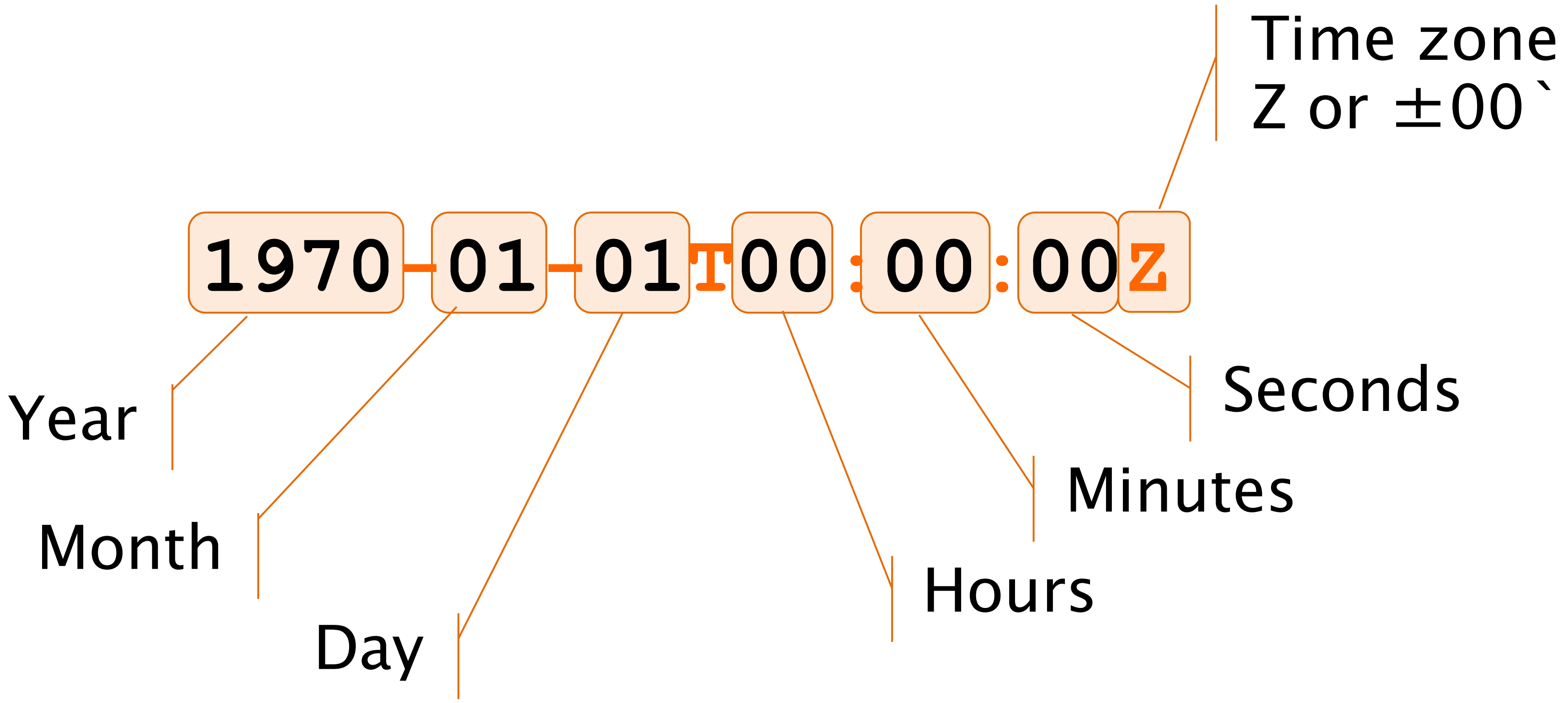
ISO-8601


PUBLIC SERVICE ANNOUNCEMENT:
OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.
THIS IS *THE* CORRECT WAY TO WRITE NUMERIC DATES:
2013-02-27
THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:
02/27/2013 02/27/13 27/02/2013 27/02/13
20130227 2013.02.27 27.02.13 27-02-13
27.2.13 2013. II. 27. 27/2-13 2013.158904109
MMXIII-II-XXVII MMXIII LVII/CCCLXV 1330300800
((3+3)×(111+1)-1)×3/3-1/3³
10/11011/1101 02/27/20/13
2 3 1 4 01237 5 67 8
HISSSSS
2-27-13

© XKCD – https://xkcd.com/1179/

Date/Time String Format

- Default format as defined by the ISO-8601standard



Time Intervals factory methods

Method	Purpose
`of()`	Creates interval from specified amount of **`TemporalUnits`**
`ofXxxx()`	Creates interval from specified amount of units (Xxxx is : **Days**, **Hours**, etc.)
`between()`	Creates interval between two temporal points

Example: Elapsed Time

```
Instant start = Instant.now();
//…
Instant end = Instant.now();
Duration elapsed =
        Duration.between(start, end);
System.out.println(elapsed);
```

```
PT2.005S
```

Testing

- Testing code that is time dependent can be difficult
- For this purpose we have class **`Clock`**
    - Can be used as argument of **`now()`**
- It represent an alternate time and date

Clock factories

- Clocks can be created with:
- `fixed(instant, zone)`
- `offset(base, offset)`
- `systemDefaultZone()`
- `systemUTC()`

Date-base computation

```
static double totalDue(double amount,
      LocalDate begin, double monthlyRate){
  LocalDate today = LocalDate.now();
```

Depends on time of test execution

```
  Period interval = Period.
                      between(begin, today);
  int months = interval.getMonths();
  double compoundRate =
          Math.pow(1.0+monthlyRate, months);
  return amount*compoundRate;
}
```

Date-base comp. testable

```
static double totalDue(double amount,
       LocalDate begin, double monthlyRate,
        Clock clock){
   LocalDate today = LocalDate.now(clock);
   Period interval = Period.
                         between(begin, today);
   int months = interval.getMonths();
   double compoundRate =
          Math.pow(1.0+monthlyRate, months);
   return amount*compoundRate;
}
```

Date-based test

```
@Test
public static void testTotalDue(){
  LocalDate begin = LocalDate.of(2025,4,10);
  LocalDate in4 = begin.plusMonths(4));
  ZoneId zone = ZoneId.systemDefault();
  Clock clock = Clock.fixed(in4
        .atStartOfDay(zone).toInstant(),zone);
  double r = 0.01;
  int amount = 1000;
  double t = totalDue(amount, begin, r, clock);
  assertEquals(amount*Math.pow(1+r, 4), t, 1);
}
```

Summary

- Old `Date` class does not handle time zones correctly
- New classes provide a consistent structure for both time and date measures:
  - They are immutable
  - Operations can be performed using existing methods