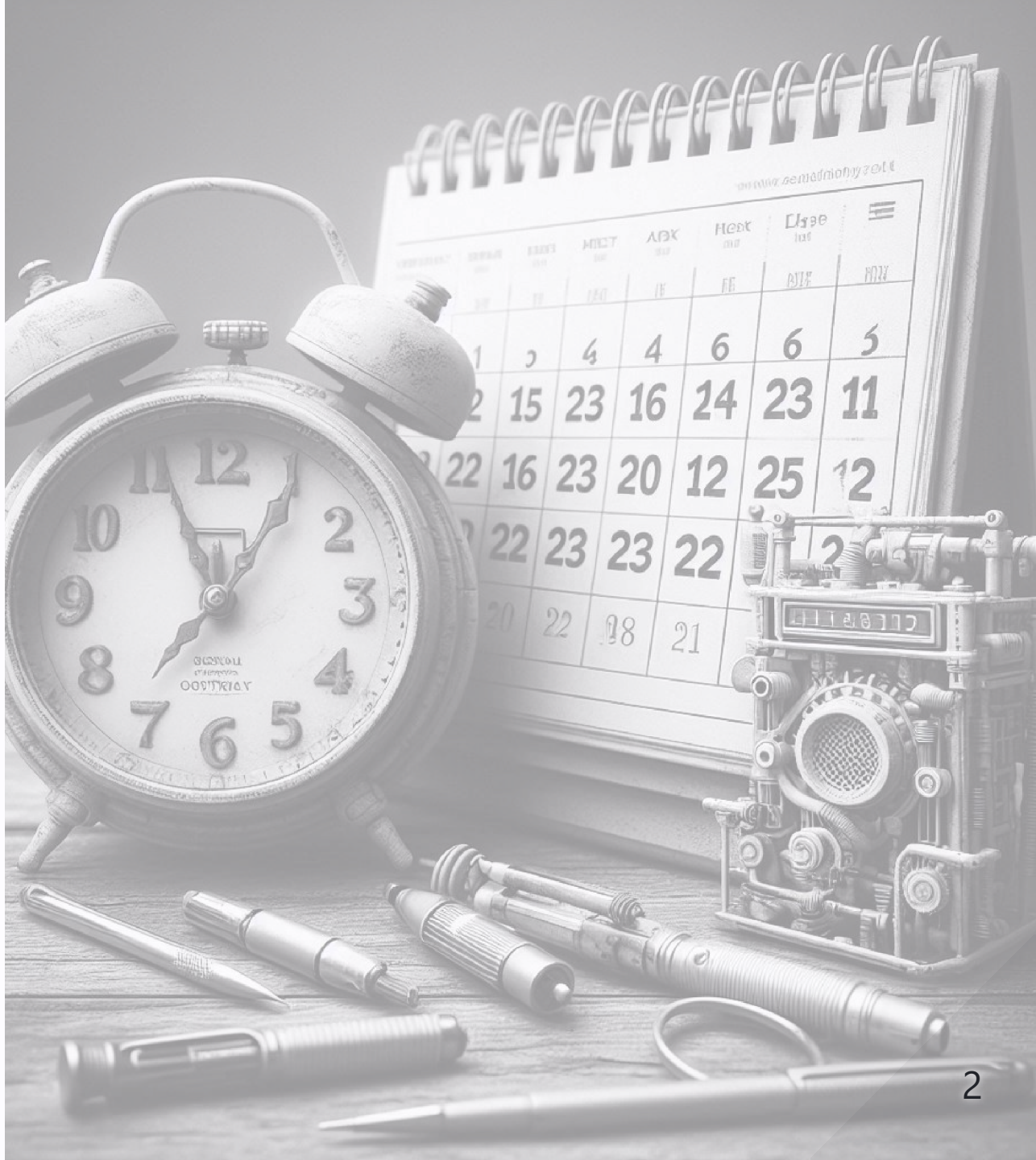
Java Date and Time API

Sualeh Fatehi

Java Date and Time API



Design Principles

- Distinguish between machine and human views
- Well-defined and clear purpose
- Immutable, thread-safe
- Reject null and bad arguments early
- Extensible, by use of strategy pattern
- Fluent interface with chained methods

Instant

- Point on a discretized timeline
- Stored to nanosecond resolution
  - `long` for seconds since epoch, and
  - `int` for nanosecond of second
- Convert to any date time field using a `Chronology`

Partial

- An indication of date or time that cannot identify a specific, unique instant
- Definition uses fields such as year, month, day of month, and time of day
- Commonly used partials, such as `LocalDate` and `LocalTime` are available
- Useful partials like `MonthDay` and `YearMonth` are also available

Duration

- Precise length of elapsed time, in nanoseconds
- Does not use date-based constructs like years, months, and days
- Can be negative, if end is before start

Period

- A length of elapsed time
- Defined using calendar fields - years, months, and days (not minutes and seconds)
- Takes time zones into account for calculation

Chronology

- Pluggable calendar system
- Provides access to date and time fields
- Built-in
  - ISO8601 (default): `IsoChronology`
  - Chinese: `MinguoChronology`
  - Japanese: `JapaneseChronology`
  - Thai Buddhist: `ThaiBuddhistChronology`
  - Islamic: `HijrahChronology`

Clock

- Gets the current instant using a time-zone
- Use instead of `System.currentTimeMillis()`
- Use an alternate clock for testing

Testable Code

```
public class SomeDomainObject {
  @Inject private Clock clock;
  public void process() {
    LocalDate date = LocalDate.now(clock);
    // ... do something with date
  }
}
```

Packages

- `java.time` - instants, durations, dates, times, time zones, periods
- `java.time.format` - formatting and parsing
- `java.time.temporal` - field, unit, or adjustment access to temporals
- `java.time.zone` – support for time zones
- `java.time.chrono` - calendar systems other than ISO-8601

Consistent Operations

- `of` - static factory, validates input
- `from` - static factory, converts to target class
- `get` - returns part of the state
- `is` - queries the state
- `with` - immutable copy with elements changed
- `to` - converts to another object type
- `plus`, `minus` - immutable copy after operation

Staying Constant

- Day of week, for example `DayOfWeek. SUNDAY`
- Month , for example

  `LocalDate.of(2014, Month. MAY , 20)`
- Time units, for example

  `Instant.now().plus(1, ChronoUnit. DAYS)`
- Other useful constants, for example
  - `LocalTime.MIDNIGHT // 00:00`
  - `LocalTime.NOON // 12:00`

Old and New

Only if you have to...

- Calendar interconversions
  - `toInstant()`
  - `toZonedDateTime()`
  - `from(ZonedDateTime)`
- Date interconversions
  - `toInstant()`
  - `from(Instant)`

Formatting

- Format with a `DateTimeFormatter` instance
- Internationalization is supported
- Custom formats can be used, including am/pm for time

Parsing

- Parse with a `DateTimeFormatter` instance
- `parse(...)` methods return a temporal
- Use `from(...)` to convert to a known date or time type

Temporal Adjusters

- Strategy for adjusting a temporal object
- Use `with(...)` to convert to another temporal object

```
LocalTime time = LocalTime.NOON;
time.with(temporal ->
            temporal.plus(4, ChronoUnit.MINUTES)));
```

Built-in Temporal Adjusters

Some useful temporal adjusters are built into `java.time.temporal.TemporalAdjusters`

- `firstDayOfMonth()`
- `firstDayOfYear()`
- `firstInMonth(DayOfWeek)`
- `next(DayOfWeek)`
- `previous(DayOfWeek)`

Temporal Queries

- Strategy for extracting information from temporals
- Externalize the process of querying
- Examples
  - get the time zone in a temporal
  - check if date is February 29 in a leap year
  - calculate days until your next birthday
- `TemporalQueries` class has implementations of common queries

Slides and Code

github.com/**sualeh/make-a-date**