1 Data Types

1.1 Date and Time

1.1.1 LocalDate

LocalDate is an immutable date-time object that represents a date, often viewed as year-month-day. Other date fields, such as day-of-year, day-of-week and week-of-year, can also be accessed.

```
1         // to obtain, e.g.
2         static LocalDate of(int year, int month, int
              dayOfMonth)
3         static LocalDate of(int year, Month month, int
              dayOfMonth)
4         static LocalDate ofInstant(Instant instant, ZoneId
              zone)
5         static LocalDate parse(CharSequence text,
              DateTimeFormatter formatter)
6
7         // instance methods, e.g.
8         LocalDateTime atTime(int hour, int minute, int second,
              int nanoOfSecond)
9         LocalDateTime atTime(LocalTime time)
10
11        int getDayOfMonth()
12        DayOfWeek getDayOfWeek()
13        int getDayOfYear()
14        Month getMonth()
15        int getMonthValue()
16        int getYear()
17
18        // same for plus
19        LocalDate minus(long amountToSubtract, TemporalUnit
              unit)
20        LocalDate minusDays(long daysToSubtract)
21        LocalDate minusMonths(long monthsToSubtract) //etc
```

1.1.2 LocalTime

LocalTime is an immutable date-time object that represents a time, often viewed as hour-minute-second. Time is represented to nanosecond precision. For example, the value "13:45.30.123456789" can be stored in a LocalTime.

```
1         // to obtain, e.g.
2         static LocalTime of(int hour, int minute, int second,
              int nanoOfSecond)
3         static LocalTime ofInstant(Instant instant, ZoneId
              zone)
4
5         // instance methods, e.g.
6         LocalDateTime atDate(LocalDate date)
7
8         int getHour()
9         int getMinute() //etc.
10
11        // same for minus
12        LocalTime plus(long amountToAdd, TemporalUnit unit)
13        LocalTime plusNanos(long nanosToAdd) // etc.
14
15        // returns copy
16        LocalTime withHour(int hour)
17        LocalTime withMinute(int minute) //etc.
```

1.1.3 LocalDateTime

```
1         // to obtain, e.g.
2         static LocalDateTime of(int year, Month month, int
              dayOfMonth, int hour, int minute, int second, int
              nanoOfSecond)
3         static LocalDateTime of(LocalDate date, LocalTime time)
4         // instance methods analogous to above
```

1.1.4 Month

In addition to the textual enum name, each month-of-year has an int value (1-12). Do not use ordinal() to obtain the numeric representation of Month. Use getValue() instead.

```
1         // to obtain, e.g.
2         static Month of(int month)
3         Month e = Month.of(10); // DECEMBER
4         static Month valueOf(String name)
5         Month m = Month.valueOf("DECEMBER"); // DECEMBER
6
7         // instance methods, e.g.
```

```
8         int getValue()
9         int length(boolean leapYear)
10        minus(long months)
11        plus(long months)
```

1.1.5 ChronoUnit

```
1         // to obtain, e.g.
2         static ChronoUnit valueOf(String name)
3
4         // instance methods, e.g.
5         <R extends Temporal> R addTo(R temporal, long amount)
              // returns a copy!
6         long between(Temporal temporal1Inclusive, Temporal
              temporal2Exclusive)
```

1.1.6 Instant

An Instant represents a specific moment in time using GMT. Consequently, there is no time zone information.

```
1         // to obtain, e.g.
2         static Instant from(TemporalAccessor temporal)
3         static Instant now()
4         static Instant ofEpochMilli(long epochMilli)
5
6         // instance methods, e.g.
7         OffsetDateTime atOffset(ZoneOffset offset)
8         ZonedDateTime atZone(ZoneId zone)
9
10        Instant minus(long amountToSubtract, TemporalUnit
              unit) //returns copy! others too
11        Instant minus(TemporalAmount amountToSubtract)
12
13        Instant minusMillis(long millisToSubtract)
14        Instant minusNanos(long nanosToSubtract)
15
16        var instant = trainDay.toInstant(); // will not
              compile if this is a LocalDateTime!
```

1.1.7 Period

This class models a quantity or amount of time in terms of years, months and days. See Duration for the time-based equivalent to this class.

Durations and periods differ in their treatment of daylight savings time when added to ZonedDateTime. A Duration will add an exact number of seconds, thus a duration of one day is always exactly 24 hours. By contrast, a Period will add a conceptual day, trying to maintain the local time.

For example, consider adding a period of one day and a duration of one day to 18:00 on the evening before a daylight savings gap. The Period will add the conceptual day and result in a ZonedDateTime at 18:00 the following day. By contrast, the Duration will add exactly 24 hours, resulting in a ZonedDateTime at 19:00 the following day (assuming a one hour DST gap).

The supported units of a period are YEARS, MONTHS and DAYS. All three fields are always present, but may be set to zero.

The period is modeled as a directed amount of time, meaning that individual parts of the period may be negative.

```
1         // to obtain, e.g.
2         static Period between(LocalDate startDateInclusive,
              LocalDate endDateExclusive)
3         static Period of(int years, int months, int days)
4         static Period ofDays(int days) // other fields will be
              0
5         static Period ofMonths(int months)
6
7         // instance methods, e.g.
8         Temporal addTo(Temporal temporal)
9         Period minusDays(long daysToSubtract) // all return
              copies!
10        minusMonths(long monthsToSubtract)
11
12        Period withMonths(int months) // copies, too!
13        Period withYears(int years)
14
15        int getDays()
```

1.1.8 Duration

This class models a quantity or amount of time in terms of seconds and nanoseconds. It can be accessed using other duration-based units, such as minutes and hours. In addition, the DAYS unit can be used and is treated as exactly equal to 24 hours, thus ignoring daylight savings effects.

See Period for the date-based equivalent to this class.

```
    // to obtain, e.g.
    static Duration of(long amount, TemporalUnit unit)
    static Duration ofDays(long days)

    // instance methods, e.g.
    Duration dividedBy(long divisor) // all these copy
    long dividedBy(Duration divisor)

    long get(TemporalUnit unit)
    int getNano()
    long getSeconds()
```

1.2 String and StringBuffer

1.2.1 String

```
    // strip()-related methods (these are the only ones)
    strip(), stripLeading(), stripTrailing(), stripIndent()

    // indent(): normalizes the output by adding a line break
       to the end
    // does not change the indentation, but still adds a
       normalizing line break
    System.out.println(phrase.indent(0).length());

    // translateEscapes()
    // these print 2 lines:
    System.out.println("cheetah\ncub");
    System.out.println("cheetah\ncub".translateEscapes());
    System.out.println("cheetah\\ncub".translateEscapes());
    - this prints 1:
    System.out.println("cheetah\\ncub");

    // format string
    var quotes = """
    \"The Quotes that Could\" // could remove both backslashes
    \"\"\"                    // could remove 2 backslashes
    """;

    // there is no reverse()
```

1.2.2 StringBuilder

```
1
2         // instance methods, e.g.
3         char charAt(int index)
4         IntStream chars()
5
6         int indexOf(String str)
7
8         int length()
9
10        StringBuilder
11        delete(int start, int end)
```

```
1         // to obtain, e.g.
2
3         // instance methods, e.g.
```

```
1         // to obtain, e.g.
2
3         // instance methods, e.g.
```