1Z0-808 Exam Topic Reviewer

TopicId: 1033
Topic: Date and Time API (java.time)

August 5, 2025

Introduction: A New Era for Dates and Times

For years, Java developers struggled with the old `java.util.Date` and `java.util.Calendar` APIs. They were mutable, had confusing method names, were not thread-safe, and used a 0-based month system. The 1Z0-808 exam will test your knowledge of the modern `java.time` API introduced in Java 8, which solves all these problems. Your mantra for this topic should be: **Immutability and Clarity**.

1 The Core Classes: Human-Readable Time

These are the main classes you'll use to represent dates and times. They are all in the `java.time` package and are timezone-agnostic.

1.1 `LocalDate`

Represents a date without time or a time zone (e.g., year-month-day).

- **Creation:**

```
LocalDate today = LocalDate.now(); // Current date from system clock
LocalDate myBirthday = LocalDate.of(1995, 8, 15); // Year, Month, Day
LocalDate parsedDate = LocalDate.parse("2025-12-25"); // From a String
```

Exam Trap: The month is 1-based (1 for January, 12 for December). Providing an invalid value like `LocalDate.of(2025, 13, 1)` will throw a `DateTimeException` at runtime.

1.2 `LocalTime`

Represents a time without a date or a time zone (e.g., hour-minute-second).

- **Creation:**

```
LocalTime now = LocalTime.now(); // Current time
LocalTime meetingTime = LocalTime.of(13, 30); // 1:30 PM
LocalTime lunchTime = LocalTime.parse("12:00:00"); // From a String
```

1.3 `LocalDateTime`

Represents a date and time combined, but still without a time zone.

- **Creation:**

```
LocalDateTime rightNow = LocalDateTime.now(); // Current date and time
LocalDateTime event = LocalDateTime.of(2025, 8, 5, 15, 30);
// You can also combine a LocalDate and LocalTime
LocalDateTime combined = LocalDateTime.of(myBirthday, meetingTime);
```

2 The Golden Rule: Immutability

Every class in the `java.time` package is **immutable**. This means once an object is created, it cannot be changed. All manipulation methods (like `plusDays`, `minusHours`) return a **new** object. Failing to assign the result of these methods is a classic mistake and a frequent exam trick.

Incorrect Usage (Common Trap):

```
LocalDate date = LocalDate.of(2025, 1, 20);
date.plusDays(10); // This line does nothing to the 'date' variable!
System.out.println(date); // Prints 2025-01-20
```

Correct Usage:

```
LocalDate date = LocalDate.of(2025, 1, 20);
date = date.plusDays(10); // Re-assign the new object to the variable
System.out.println(date); // Prints 2025-01-30
```

3 Representing Spans of Time: `Period` vs. `Duration`

The exam will test you on the difference between these two classes. The key is what kind of temporal unit they represent.

3.1 `Period`: Date-Based Amounts

Represents a quantity of time in terms of **years, months, and days**. It's used with `LocalDate` and `LocalDateTime`.

- **Creation:** `Period p = Period.of(2, 3, 10);` // 2 years, 3 months, 10 days
- **Usage:** `LocalDate futureDate = today.plus(p);`
- **Exam Trap:** The `ofYears`, `ofMonths`, `ofDays` factory methods are static and do NOT chain. The last one called wins.

  ```
  // This creates a period of 5 DAYS ONLY, not 1 year and 5 days!
  Period trickyPeriod = Period.ofYears(1).ofDays(5);

  // To create 1 year and 5 days, you must use:
  Period correctPeriod = Period.of(1, 0, 5);
  ```

3.2 `Duration`: Time-Based Amounts

Represents a quantity of time in terms of **hours, minutes, seconds, and nanoseconds**. It's used with `LocalTime` and `LocalDateTime`.

- **Creation:** `Duration d = Duration.ofHours(3).plusMinutes(30);`
- **Usage:** `LocalDateTime later = rightNow.plus(d);`

- **Exam Trap:** You cannot use a `Duration` with a `LocalDate`. A `LocalDate` does not have a time component. The following code throws an `UnsupportedTemporalTypeEx`

```
LocalDate date = LocalDate.now();
Duration d = Duration.ofHours(24);
date.plus(d); // Throws Exception at runtime!
```

4 Formatting and Parsing

To convert `java.time` objects to and from Strings, you use `DateTimeFormatter`.

- **Formatting (Object -¿ String):**

```
LocalDateTime event = LocalDateTime.of(2025, 8, 5, 15, 0);

// Using a predefined standard formatter
String iso = event.format(DateTimeFormatter.ISO_DATE_TIME);
// Result: "2025-08-05T15:00:00"

// Using a custom pattern
DateTimeFormatter formatter =
    DateTimeFormatter.ofPattern("MMMM dd, yyyy 'at' hh:mm a");
String custom = event.format(formatter);
// Result: "August 05, 2025 at 03:00 PM"
```

- **Parsing (String -¿ Object):**

```
String input = "May 20, 2026";
DateTimeFormatter f = DateTimeFormatter.ofPattern("MMMM d, yyyy");
LocalDate parsed = LocalDate.parse(input, f);
System.out.println(parsed); // Prints 2026-05-20
```

Key Takeaways for the 1Z0-808 Exam

- **Immutability is everything.** Manipulation methods create and return new instances. Check if the result is assigned!
- Know the core classes: `LocalDate`, `LocalTime`, `LocalDateTime`.
- Understand the difference: `Period` is for date units (Y, M, D) and works with `LocalDate`. `Duration` is for time units (H, M, S) and works with `LocalTime`.
- Be wary of creating objects with invalid values (e.g., month 13)—this results in a runtime `DateTimeException`.
- Remember that `Period.ofYears(1).ofMonths(5)` creates a period of 5 months, not 1 year and 5 months.

- Know how to use `DateTimeFormatter` for both formatting (`format()`) and parsing (`parse()`).