Introduction to Common Built-in Functions - Date and Time

Concepts Explained:

Date and time are essential in many applications. From showing the current date on a website to scheduling events, handling dates and times is a common task in programming.

In Java, we can manage date and time using built-in functions. Older versions of Java had some issues with how they handled dates and times, but newer versions (from Java 8 onwards) introduced a much better way to work with them.

1. **Legacy Date and Time API (Before Java 8):**
   - **`java.util.Date` and `java.util.Calendar`:**
     - These were the main classes for handling date and time in earlier versions of Java.
     - Problems: They were mutable, not thread-safe, and lacked proper timezone handling.
2. **Modern Java Date and Time API (Java 8 onwards):**
   - Introduced in Java 8, the **`java.time`** package offers improved date and time handling.
   - **Key Improvements**:
     - Immutability and thread-safety.
     - Clear separation of date and time into specific classes.
     - Extensive support for formatting, parsing, and manipulating dates and times.
3. **Key Classes in the Modern Date-Time API:**
   - **`LocalDate`**: Represents a date (year, month, day) without time or timezone information.
     - Example: `LocalDate.now()` gets the current date.
   - **`LocalTime`**: Represents time (hours, minutes, seconds) without a date or timezone.
     - Example: `LocalTime.now()` gets the current time.
   - **`LocalDateTime`**: Combines both date and time, but still without timezone information.
     - Example: `LocalDateTime.now()` returns the current date and time.
   - **`ZonedDateTime`**: Represents date and time with a specific timezone.
     - Example: `ZonedDateTime.now()` returns current date, time, and timezone.
4. **Date-Time Operations:**
   - **Parsing and Formatting**:
     - Use **`DateTimeFormatter`** to convert between `String` and date

objects, or to format date/time objects into readable strings.

- **Arithmetic Operations**:
    - Methods like `plusDays()`, `minusMonths()` can be used to add or subtract time from a date or time.
- **Date Comparison**:
    - Use `isBefore()`, `isAfter()`, and `isEqual()` to compare dates.
- **Time Differences**:
    - Use **`ChronoUnit`** to calculate the difference between two dates or times in units such as days, months, or years.

5. **Thread-Safety and Immutability**:
    - Classes in the `java.time` package are immutable, meaning once created, their state cannot change, making them thread-safe.

Learning Problem Statement (with Solution):

Problem Statement: Create a Java program that:

- Takes a date input from the user (in the format `dd-MM-yyyy`).
- Displays the day of the week for the given date.
- Calculates and shows the number of days between the given date and the current date.
- Displays the current date and time in a formatted manner.

Solution:

Step-by-Step Breakdown:

1. **Input Date Parsing**: We will take a `String` date input in the format `dd-MM-yyyy` and parse it into a `LocalDate` object using `DateTimeFormatter`.
2. **Day of the Week**: Extract the day of the week using `LocalDate.getDayOfWeek()`. 3. **Calculate Difference**: Use `ChronoUnit.DAYS.between()` to calculate the number of days between the input date and the current date.
4. **Format Current Date/Time**: Display the current date and time using `DateTimeFormatter`.

Code Example without Function/Method Implementation

```
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.Scanner;

public class DateTimeExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Step 1: Input date in dd-MM-yyyy format
        System.out.println("Enter a date (dd-MM-yyyy): ");
        String inputDate = scanner.nextLine();

        // Step 2: Parse input date using DateTimeFormatter
         DateTimeFormatter formatter =
     DateTimeFormatter.ofPattern("dd-MM-yyyy");
        LocalDate date = LocalDate.parse(inputDate, formatter);

        // Step 3: Find the day of the week
        DayOfWeek dayOfWeek = date.getDayOfWeek();
        System.out.println("Day of the Week: " + dayOfWeek);

     // Step 4: Calculate the difference between input date and
current date
        LocalDate currentDate = LocalDate.now();
        long daysBetween = ChronoUnit.DAYS.between(date, currentDate);
System.out.println("Days between input date and current date: " +
daysBetween);

       // Step 5: Display the current date and time in a formatted way
        LocalDateTime currentDateTime = LocalDateTime.now();
        DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
        System.out.println("Current Date and Time: " +
currentDateTime.format(dateTimeFormatter));
    }
}
```

Code Example with Function/Method Implementation

```
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.Scanner;

public class DateTimeExample {

    // Main method - entry point of the program
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Step 1: Get user input date in dd-MM-yyyy format
        String inputDate = getInputDate(scanner);

        // Step 2: Parse the input date to LocalDate
        LocalDate parsedDate = parseDate(inputDate);

        // Step 3: Find and display the day of the week for the input
date
        displayDayOfWeek(parsedDate);

        // Step 4: Calculate and display the number of days between
input date and current date
        displayDaysBetween(parsedDate);

        // Step 5: Display the current date and time in a formatted
way
        displayCurrentDateTime();
    }

    // Step 1: Function to get user input
    public static String getInputDate(Scanner scanner) {
        System.out.println("Enter a date (dd-MM-yyyy): ");
        // Read and return the date string entered by the user
        return scanner.nextLine();
    }
    // Step 2: Function to parse the input date
    public static LocalDate parseDate(String inputDate) {
        // Define the date format to be used for parsing
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd-MM-yyyy");
        // Parse the date string into a LocalDate object using the
formatter
        return LocalDate.parse(inputDate, formatter);
    }

    // Step 3: Function to display the day of the week for the input
date
    public static void displayDayOfWeek(LocalDate date) {
        // Get the day of the week for the parsed date
        DayOfWeek dayOfWeek = date.getDayOfWeek();
        // Print the day of the week
        System.out.println("Day of the Week: " + dayOfWeek);
    }

    // Step 4: Function to display the number of days between the
input date and the current date
    public static void displayDaysBetween(LocalDate date) {
        // Get the current date
        LocalDate currentDate = LocalDate.now();
        // Calculate the number of days between the input date and the
current date
        long daysBetween = ChronoUnit.DAYS.between(date, currentDate);
        // Print the number of days between the two dates
        System.out.println("Days between input date and current date:
" + daysBetween);
    }

    // Step 5: Function to display the current date and time in a
formatted way
    public static void displayCurrentDateTime() {
        // Get the current date and time
        LocalDateTime currentDateTime = LocalDateTime.now();
        // Define the format to display the date and time
        DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
        // Print the current date and time in the specified format
        System.out.println("Current Date and Time: " +
currentDateTime.format(dateTimeFormatter));
    }
}
```

Explanation of Key Parts:

- **`LocalDate.parse()`**: Converts a string into a `LocalDate` object.
- **`getDayOfWeek()`**: Retrieves the day of the week for the given date.
- **`ChronoUnit.DAYS.between()`**: Calculates the difference in days between two dates.
- **`DateTimeFormatter`**: Formats the date and time into a readable string.