Date Functions in java:

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

import java.util.Date;

import java.text.SimpleDateFormat;

public class DateFunctions {

public static void main(String[] args) {

// Using java.util.Date

Date currentDate = new Date();

System.out.println("Current Date (java.util.Date): " + currentDate);

// Using java.time classes

LocalDate localDate = LocalDate.now();

System.out.println("Current Date (LocalDate): " + localDate);

LocalTime localTime = LocalTime.now();

System.out.println("Current Time (LocalTime): " + localTime);

LocalDateTime localDateTime = LocalDateTime.now();

System.out.println("Current Date and Time (LocalDateTime): " + localDateTime);

// Formatting with DateTimeFormatter

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

String formattedDateTime = localDateTime.format(formatter);

System.out.println("Formatted Date and Time: " + formattedDateTime);

// Using SimpleDateFormat

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

String formattedDate = sdf.format(currentDate);

System.out.println("Formatted Date (SimpleDateFormat): " + formattedDate);

}

}

Java provides several classes for working with dates and times. Here's a breakdown of key classes and methods:

1. java.util.Date:

* Represents a specific instant in time, measured in milliseconds since January 1, 1970, 00:00:00 GMT.
* **Constructors:**
  + Date(): Creates a Date object representing the current time.
  + Date(long milliseconds): Creates a Date object representing the time specified by the given milliseconds value.
* **Methods:**
  + getTime(): Returns the number of milliseconds since the epoch.
  + setTime(long time): Sets the Date object to a specific point in time.
  + before(Date when): Checks if the date is before the specified date.
  + after(Date when): Checks if the date is after the specified date.
  + compareTo(Date anotherDate): Compares two dates for ordering.
  + equals(Object obj): Checks if two dates are equal.
  + clone(): Creates a copy of the Date object.
  + toInstant(): Converts the Date object to an Instant object. (Java 8 and later)
* **Note:** Many methods in java.util.Date are deprecated, and the newer java.time package is recommended for most use cases.

2. java.time Package (Java 8 and later):

* Introduced to provide a more robust and user-friendly API for date and time manipulation.
* Key classes include:
  + LocalDate: Represents a date (year, month, day) without time or time zone.
    - now(): Creates a LocalDate object representing the current date.
    - of(int year, int month, int dayOfMonth): Creates a LocalDate object with the specified date.
  + LocalTime: Represents a time (hour, minute, second, nanosecond) without a date or time zone.
    - now(): Creates a LocalTime object representing the current time.
    - of(int hour, int minute, int second): Creates a LocalTime object with the specified time.
  + LocalDateTime: Represents a date and time without a time zone.
    - now(): Creates a LocalDateTime object representing the current date and time.
    - of(int year, int month, int dayOfMonth, int hour, int minute, int second): Creates a LocalDateTime object with the specified date and time.
  + DateTimeFormatter: Used for formatting and parsing date and time objects.
    - ofPattern(String pattern): Creates a DateTimeFormatter with a specified pattern (e.g., "yyyy-MM-dd", "HH:mm:ss").
    - format(TemporalAccessor): Formats a date/time object into a string.
    - parse(CharSequence, TemporalQuery): Parses a string into a date/time object.

3. java.text.SimpleDateFormat:

* Used for formatting and parsing dates in a specific format.
* **Constructor:** SimpleDateFormat(String pattern): Creates a SimpleDateFormat object with the specified pattern.
* **Methods:**
  + format(Date date): Formats a Date object into a string.
  + parse(String source): Parses a string into a Date object.