Tab 1

Meta title: Calculate the difference between two dates in Java.
Meta description: Learn the different methods to calculate the difference between any two Java date instances by different methods and their pros and cons.

https://chatgpt.com/share/680ce43d-c5cc-800e-8131-235251ac66df
Done

Calculating the difference between two Java date instances

Java has many ways to get the time difference between two dates, from simple methods like using the Date class to advanced methods. The Java standard library and third-party libraries provide classes and methods to calculate the months between two dates.

In this blog, we will learn in detail about the different methods that can be used to calculate the difference between any two dates.

Table of contents:

Java Date Difference Calculation

In Java, dates are useful for knowing the time between any two events. For example, you want to find the difference between an employee's joining date, a student's admission date, or an appointment date. Java offers many ways to calculate this. You can use various methods, but the method you will choose depends on how you need the result to be, e.g., milliseconds, seconds, minutes, or days.

Methods of Calculating the difference between two Java date instances in Java

In Java, there are several ways to calculate the difference between two Dates or times. Some of them are:

Method 1: Using SimpleDateFormat and Date Class in Java

This method parses the date string into a Date object using **SimpleDateFormat** and then simply calculates the difference between the **getTime()** values of the two Date instances. SimpleDateFormat is used to define the date format and parse the date string into Date objects

Steps of using SimpleDateFormat and Date Class:

1. **Parsing Date Strings:** By using the **SimpleDateFormat,** you can parse the date string(yyyy-MM-dd HH:mm:ss) into Date objects.
2. **Subtracting Dates:** Once you have the Date objects, you can get the time in milliseconds using the **getTime()** method.
3. **Calculating the Difference:** Subtract the above two time values and then convert them into days, hours, minutes, or seconds.

Example:

[codelab lang="java"]<pre>

```
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateDifferenceMethod1 {
    public static void main(String[] args) {
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            // Define two date strings (in this case, "yyyy-MM-dd HH:mm:ss" format)
            String dateStr1 = "2025-03-23 10:00:00";  // Older date
            String dateStr2 = "2025-03-25 12:30:00";  // Newer date
            Date date1 = sdf.parse(dateStr1);
            Date date2 = sdf.parse(dateStr2);
            // Calculate the difference in milliseconds
            long diffInMillis = date2.getTime() - date1.getTime();
            long diffInSeconds = diffInMillis / 1000;
            long diffInMinutes = diffInSeconds / 60;
            long diffInHours = diffInMinutes / 60;
            long diffInDays = diffInHours / 24;
            System.out.println("Difference in Days: " + diffInDays);
            System.out.println("Difference in Hours: " + diffInHours);
            System.out.println("Difference in Minutes: " + diffInMinutes);
            System.out.println("Difference in Seconds: " + diffInSeconds);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    }
</pre>[/codelab]
```

Output:

```
Difference in Days: 2
Difference in Hours: 50
Difference in Minutes: 3030
Difference in Seconds: 181800
```

Explanation:
The above program finds the difference between the two dates in days, hours, minutes, and seconds. It parses the date strings into Date objects, gets the time difference in milliseconds, and then converts it into various time units.

Method 2: Using TimeUnit Class in Java

The **TimeUnit class in** Java is part of the java.util.concurrent package. It is used to convert the time differences between many units like milliseconds, seconds, minutes, hours, etc.. In this class, no manual calculation is required. It is part of the java.util.concurrent package and is used to make changes like milliseconds to seconds, minutes, hours, or days.

Example:

```
[codelab lang="java" run="disable"]<pre>
import java.util.Date;
import java.util.concurrent.TimeUnit;
public class DateDifferenceMethod2 {
    public static void main(String[] args) {
        Date d1= new Date();
        try {
            // Sleep for 20 seconds
            Thread.sleep(20000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Date d2= new Date();
        // difference
        long  Millis = d2.getTime() - d1.getTime();
        long  Hours = TimeUnit.MILLISECONDS.toHours(Millis);
        long  Minutes = TimeUnit.MILLISECONDS.toMinutes(Millis);
```

```
        long Seconds = TimeUnit.MILLISECONDS.toSeconds(Millis);
        System.out.println("Difference in Hours: " + Hours);
        System.out.println("Difference in Minutes: " + Minutes);
        System.out.println("Difference in Seconds: " + Seconds);
    }
}
</pre>[/codelab]
```

Output:

```
Difference in Hours: 0
Difference in Minutes: 0
Difference in Seconds: 20
```

Explanation:
The above Java code takes two times before and after a 20 second gap, then it calculates the time difference in milliseconds, and converts it into hours, minutes, and seconds using the **TimeUnit** method**.**

Method 3: Using LocalDate and ChronoUnit Class in Java

In Java 8 and later, the java.time package gives easy usage of the dates and times with classes like **LocalDate** for the date without time and LocalDateTime with date and time. The **ChronoUnit** class is used to calculate the difference between two LocalDate or LocalDateTime objects easily. It can be used with both LocalDate and LocalDateTime objects.

Steps of Using LocalDate and ChronoUnit Class in Java:

1. Use LocalDate or LocalDateTime: The LocalDate class defines a date without a time, and LocalDateTime includes both the date and time.
2. Calculate the Difference: Use the **ChronoUnit.DAYS.between()** method to calculate the difference in the various units.

Example:
[codelab lang="java"]<pre>
```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
public class DateDifferenceMethod3 {
    public static void main(String[] args) {
        LocalDate d1 = LocalDate.of(2025, 3, 23);
        LocalDate d2 = LocalDate.of(2025, 3, 25);
        // Calculating the difference
        long diff = ChronoUnit.DAYS.between(d1, d2);
          System.out.println("Difference in Days: " + diff);
    }
```

}
</pre>[/codelab]

Output:

```
Difference in Days: 2
```

Explanation:
The above Java program uses LocalDate and ChronoUnit.DAYS to find the day difference between two dates. It does not make changes in the units and finds the difference in days between them.

Method 4: Using Period Class (For Date Difference in Years, Months, and Days) in Java

The Period class in Java is used for calculating the difference between two **LocalDate** objects of years, months, and days. This is helpful when you are using dates that use years and months. To use the **Period.between()** method, two LocalDate objects are needed, i.e., the **Start Date** and the **End Date.** It calculates the difference between the two dates and then breaks it into three i.e., in years, months, and days.

Note: You should be careful when you are using the dates. If the start date comes after the end date, the result will be negative for years, months, and days. Negative values are used for overdue checks or validating the date inputs.

Steps of Using Period Class:

1. **Using LocalDate:** Create two LocalDate objects, date1 and date2
2. **Calculating the Period:** Use the **Period.between()** method to get the difference in years, months, and days.

Example:
[codelab lang="java"]<pre>
import java.time.LocalDate;
import java.time.Period;
public class DateDifferenceMethod4 {
    public static void main(String[] args) {
        // Create two LocalDate instances
        LocalDate d1 = LocalDate.of(2025, 3, 23); // 2025-03-23
        LocalDate d2 = LocalDate.of(2027, 5, 25); // 2027-05-25
                // Calculate the period between the two dates
        Period period = Period.between(d1, d2);
                // Output the result (years, months, and days)
        System.out.println("Difference: " + period.getYears() + " years, " +
                        period.getMonths() + " months, " + period.getDays() + " days.");

```
    }
}
</pre>[/codelab]
```

Output:

```
Difference: 2 years, 2 months, 2 days.
```

Explanation:

The above Java program uses the LocalDate and Period to calculate the difference between two dates in years, months, and days. Two LocalDate instances are created, and then the difference with the help of Period.between() method is calculated.

Method 5: Using the Duration.between() in Java

The Duration.between() method is a part of the java.time package, which is a modern API for using the date and time. This method finds the difference between the two Temporal objects like LocalDateTime, Instant, or ZonedDateTime and gives the result as a Duration object, which shows a time difference in seconds and nanoseconds.

But like Period.between(), which is used to calculate differences in terms of years, months, and days, the Duration.between() method is used for calculating the difference in terms of time units such as seconds, minutes, hours, and milliseconds.

Note: Duration works with LocalDateTime, Instant, and ZonedDateTime objects. It does not work with LocalDate.

Syntax:

```
<pre>
Duration duration = Duration.between(startDateTime, endDateTime);
</pre>
```

Example:

```
[codelab lang="java"]<pre>
import java.time.Duration;
import java.time.LocalDateTime;
public class DurationExample {
    public static void main(String[] args) {
        LocalDateTime s = LocalDateTime.of(2025, 3, 23, 10, 0, 0);
        LocalDateTime e = LocalDateTime.of(2025, 3, 25, 12, 30, 0);
        Duration duration = Duration.between(s, e);
        // Get the difference in various time units
        long Hours = duration.toHours();  // Total hours
```

```
        long Minutes = duration.toMinutes();  // Total minutes
        long Seconds = duration.getSeconds();  // Total seconds
        System.out.println("Difference in Hours: " + Hours);
        System.out.println("Difference in Minutes: " +Minutes);
        System.out.println("Difference in Seconds: " + Seconds);
    }
}
</pre>[/codelab]
```

Output:

```
Difference in Hours: 50
Difference in Minutes: 3030
Difference in Seconds: 181800
```

Explanation:
The above code finds the time difference between two LocalDateTime values, i.e. start and end. Then it uses Duration.between() to calculate the difference and then gives the result in hours, minutes, and seconds.

Method 6: Using the JodaTime Library in Java

Joda-Time is a library that is used for date and time operations in Java. It was earlier used in older Java projects because it was simple and easy to use. You can use Joda-Time to calculate the difference between two dates or times in a flexible manner than the old Date class as it handles the time zones and formatting in an efficient manner.

Steps to Use Joda-Time for Date Difference:

Step 1. Adding Joda-Time to the Project:
First, include the Joda-Time library in your project. If you are using Maven, add the following dependency to your pom.xml:

<pre>
<dependency>
    <groupId>joda-time</groupId>
    <artifactId>joda-time</artifactId>
    <version>2.10.10</version> <!--use the latest version -->
</dependency>
</pre>

Step 2. Creating DateTime Objects:
Use the DateTime class from Joda-Time to create your dates and times.

Step 3. Calculating the Difference:
Now, use the Days, Hours, Minutes, or Seconds class to find the difference as per the required unit.

Example:
[codelab lang="java" run="disable"]<pre>
import org.joda.time.DateTime;
import org.joda.time.Days;
public class JodaExample {
    public static void main(String[] args) {
        DateTime s = new DateTime(2025, 3, 23, 10, 0, 0); // Start date:
        DateTime e = new DateTime(2025, 3, 25, 12, 30, 0);   // End date:
        // Calculate the difference in days
        int diff = Days.daysBetween(s, e).getDays();
        System.out.println("Difference in Days: " + diff);
    }
}
</pre>[/codelab]

Output:

```
Output:

Difference in Days: 2
```

Explanation:
The above Java code uses the Joda-Time library for calculating the difference in days between the two dates. It creates two DateTime objects for the start and the end dates. Then the **Days.daysBetween(start, end)** method is used to find the number of days between the two dates.

Note: Joda-Time is not used now.

Method 7: Using the Temporal#until() in Java

The **Temporal#until() method** in Java is used with Temporal objects like LocalDate, LocalDateTime, or ZonedDateTime. It calculates the time between the two Temporal objects in different units such as days, hours, months, etc.

This method is a part of the Temporal interface in Java. It allows you to find the difference between the two Temporal objects, like ChronoUnit.DAYS, ChronoUnit.HOURS, ChronoUnit.MONTHS, and more.

Syntax:
<pre>

long until(Temporal endExclusive, TemporalUnit unit);
</pre>

where:
endExclusive is The Temporal object representing the end point,
Unit is the unit of time

Example:
[codelab lang="java"]<pre>
```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZonedDateTime;
import java.time.temporal.ChronoUnit;
import java.time.ZoneId;
public class TemporalUntilCombinedExample {
    public static void main(String[] args) {
        LocalDate s = LocalDate.of(2025, 3, 23);
        LocalDate e = LocalDate.of(2025, 3, 25);
        long dBetween = s.until(e, ChronoUnit.DAYS);
        System.out.println("Days between (LocalDate): " + dBetween);
        LocalDateTime sT = LocalDateTime.of(2025, 3, 23, 10, 0);
        LocalDateTime eT = LocalDateTime.of(2025, 3, 23, 14, 0);
        long hours = sT.until(eT, ChronoUnit.HOURS);
        System.out.println("Hours between (LocalDateTime): " + hours);
        ZonedDateTime startZonedDateTime = ZonedDateTime.of(2025, 3, 23, 10, 0, 0, 0,
ZoneId.of("UTC"));
        ZonedDateTime endZonedDateTime = ZonedDateTime.of(2025, 3, 23, 10, 30, 0, 0,
ZoneId.of("UTC"));
        long Between = startZonedDateTime.until(endZonedDateTime, ChronoUnit.MINUTES);
        System.out.println("Minutes between (ZonedDateTime): " + Between);
        LocalDateTime startTimeWithSeconds = LocalDateTime.of(2025, 3, 23, 10, 0, 0);
        LocalDateTime endTimeWithSeconds = LocalDateTime.of(2025, 3, 23, 10, 5, 30);
        long secondsBetween = startTimeWithSeconds.until(endTimeWithSeconds,
ChronoUnit.SECONDS);
        System.out.println("Seconds between (LocalDateTime): " + secondsBetween);
        ZonedDateTime startZonedDateTimeWithZone = ZonedDateTime.of(2025, 3, 23, 10, 0, 0,
0, ZoneId.of("America/New_York"));
        ZonedDateTime endZonedDateTimeWithZone = ZonedDateTime.of(2025, 3, 23, 12, 30, 0,
0, ZoneId.of("America/New_York"));
        long hoursBetweenWithZone =
startZonedDateTimeWithZone.until(endZonedDateTimeWithZone, ChronoUnit.HOURS);
        System.out.println("Hours between (ZonedDateTime with time zone): " +
hoursBetweenWithZone);
        LocalDate startMonthDate = LocalDate.of(2025, 1, 1);
```

```
        LocalDate endMonthDate = LocalDate.of(2025, 3, 1);
        long monthsBetween = startMonthDate.until(endMonthDate, ChronoUnit.MONTHS);
        System.out.println("Months between (LocalDate): " + monthsBetween);
    }
}
</pre>[/codelab]
```

Output:

```
Days between (LocalDate): 2
Hours between (LocalDateTime): 4
Minutes between (ZonedDateTime): 30
Seconds between (LocalDateTime): 330
Hours between (ZonedDateTime with time zone): 2
Months between (LocalDate): 2
```

Explanation:
In the above Java code, the **until()** method is used to calculate the difference between the two dates or times. It calculates the days between two LocalDate objects, the hours between LocalDateTime objects, and the minutes between ZonedDateTime objects. It also calculates seconds and hours with time zones and months between the LocalDate objects.

Pros and Cons of Various Java Date Difference Calculation Methods

Method	Advantages	Disadvantages
Method 1: SimpleDateFormat and Date Class	It is simple to understand and easy to implement. It works in older versions of Java.	It requires manual conversion for time units. It does not handle time zones efficiently.
Method 2: Using TimeUnit Class	It makes the code more readable. It prevents rounding errors and easily converts milliseconds to days, hours, minutes, and seconds.	It cannot calculate differences in months or years.
Method 3: Using LocalDate and ChronoUnit Class	It is simple to use and has clean syntax. It easily handles differences in days, months, and years.	It only works with LocalDate, not LocalDateTime.

Method 4: Using Period Class	It is ideal for calculating differences in years, months, and days. It provides more human-readable results.	It cannot calculate differences in hours, minutes, or seconds. It only works with LocalDate.
Method 5: Using Duration.between()	It is best for calculating differences in hours, minutes, seconds, and nanoseconds. It works well with LocalDateTime, Instant, and ZonedDateTime.	It cannot directly calculate differences in days, months, or years. It requires conversion for LocalDate.
Method 6: Using Joda-Time Library	It is more flexible than Date and Calendar. It provides better time zone handling and supports both date and time difference calculations.	It adds an extra dependency to the project.
Method 7: Using Temporal#until()	It works with LocalDate, LocalDateTime, and ZonedDateTime. It is modern and efficient.	It has a more complex syntax and is less commonly used.

Conclusion

There are many ways to calculate the difference between two dates in Java. Simple methods like SimpleDateFormat and the Date class have simple calculations, and TimeUnit offers conversions. Recent Java methods like LocalDate, ChronoUnit, and Period have simple calculations for dates. Duration.between() method is used to calculate time differences, and the Temporal#until() method is used to calculate differences in any unit other than time. The data is handled in an easy manner using Joda-Time libraries.

If you want to learn more about Java, you can refer to our Java Course.

Calculating the difference between two Java date instances - FAQs

Q1. How can the difference between two datetime instances be calculated?
To get the difference between two dates, subtract date2 from date1.

Q2. How to get the number of days difference between two dates in Java?
If we want to calculate the difference in days, the ChronoUnit.DAYS.between() method is the best way

Q3. What is greater than date comparison?

"Greater than" and "Less than" operators compare the chronological order of the two dates

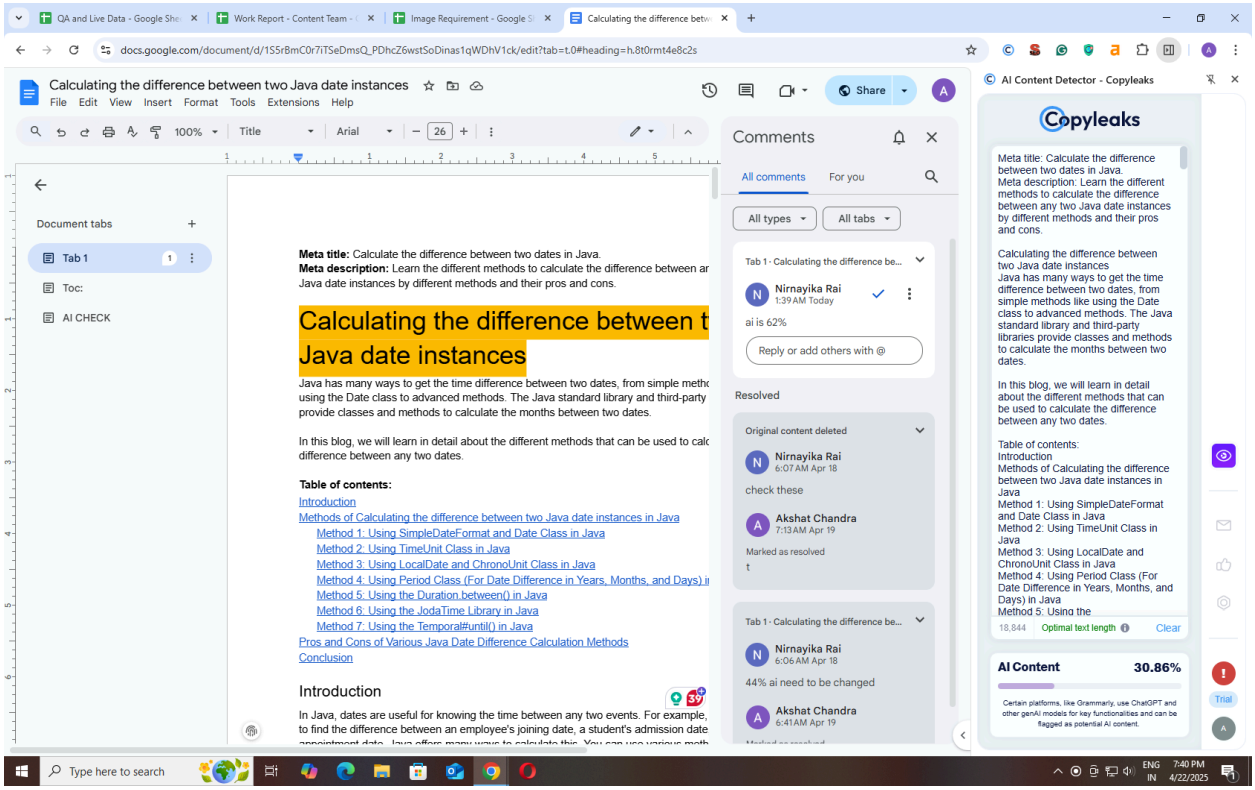
Q4. How to check if dates overlap in Java?
The **overlaps()** method takes two Interval objects as parameters and returns true if there is any intersection between the two intervals.

Q5. How to check if two dates are equal in Java?
The .equals() method of the Date class compares the current Date object to the specified object.

Toc:

Toc:

AI CHECK


Calculating the difference between two Java date instances
File Edit View Insert Format Tools Extensions Help
Share
Document tabs
Tab 1
Toc:
AI CHECK
Meta title: Calculate the difference between two dates in Java.
Meta description: Learn the different methods to calculate the difference between ar
Java date instances by different methods and their pros and cons.
Calculating the difference between t
Java date instances
Java has many ways to get the time difference between two dates, from simple metho
using the Date class to advanced methods. The Java standard library and third-party
provide classes and methods to calculate the months between two dates.
In this blog, we will learn in detail about the different methods that can be used to calc
difference between any two dates.
Table of contents:
Introduction
Methods of Calculating the difference between two Java date instances in Java
Method 1: Using SimpleDateFormat and Date Class in Java
Method 2: Using TimeUnit Class in Java
Method 3: Using LocalDate and ChronoUnit Class in Java
Method 4: Using Period Class (For Date Difference in Years, Months, and Days) i
Method 5: Using the Duration.between() in Java
Method 6: Using the JodaTime Library in Java
Method 7: Using the Temporal#until() in Java
Pros and Cons of Various Java Date Difference Calculation Methods
Conclusion
Introduction
In Java, dates are useful for knowing the time between any two events. For example,
to find the difference between an employee's joining date, a student's admission date
Comments
All comments
For you
All types
All tabs
Tab 1 · Calculating the difference be...
Nirnayika Rai
1:39 AM Today
ai is 62%
Reply or add others with @
Resolved
Original content deleted
Nirnayika Rai
6:07 AM Apr 18
check these
Akshat Chandra
7:13 AM Apr 19
Marked as resolved
t
Tab 1 · Calculating the difference be...
Nirnayika Rai
6:06 AM Apr 18
44% ai need to be changed
Akshat Chandra
6:41 AM Apr 19
AI Content Detector - Copyleaks
Copyleaks
Meta title: Calculate the difference between two dates in Java.
Meta description: Learn the different methods to calculate the difference between any two Java date instances by different methods and their pros and cons.
Calculating the difference between two Java date instances
Java has many ways to get the time difference between two dates, from simple methods like using the Date class to advanced methods. The Java standard library and third-party libraries provide classes and methods to calculate the months between two dates.
In this blog, we will learn in detail about the different methods that can be used to calculate the difference between any two dates.
Table of contents:
Introduction
Methods of Calculating the difference between two Java date instances in Java
Method 1: Using SimpleDateFormat and Date Class in Java
Method 2: Using TimeUnit Class in Java
Method 3: Using LocalDate and ChronoUnit Class in Java
Method 4: Using Period Class (For Date Difference in Years, Months, and Days) in Java
Method 5: Using the
18,844
Optimal text length
Clear
AI Content
30.86%
Certain platforms, like Grammarly, use ChatGPT and other genAI models for key functionalities and can be flagged as potential AI content.
Trial
Type here to search
ENG
IN
7:40 PM
4/22/2025

CHATGPT REPORT:

CHATGPT REPORT:

https://chatgpt.com/c/680cd529-f200-8003-a963-54079eeaf653