

**Lab Manual- Java DateTime & Lambda Expression**

**Prepared for**:

**Date:** 18th June 2024

**Prepared by:**

Document Name: Lab Manual **Document Number** SDLab333

**Contributor:**

Contents

[1. Date and Time in Java 3](#_Toc172463761)

[1.1 Create Date and Time Example Class 3](#_Toc172463762)

[1.2 Explanation: 3](#_Toc172463763)

[1.3 Execute of the Program 4](#_Toc172463764)

[2. Lambda Expressions and Functional Interfaces 4](#_Toc172463765)

[2.1 Functional Interface Analogy: 4](#_Toc172463766)

[2.2 Lambda Expression Analogy: 4](#_Toc172463767)

[2.3 Putting It Together: 5](#_Toc172463768)

[2.4 Putting It Together: 6](#_Toc172463769)

[1. Explanation: 6](#_Toc172463770)

# Date and Time in Java

Here's a simple example of how to work with date and time in Java using the **java.time package**, which is part of the Java Standard Library since Java 8.

### Create Date and Time Example Class

This example demonstrates how to **get the current date and time**, **format it**, and **parse a date string**.

**DateTimeExample.java**

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class DateTimeExample {

    public static void main(String[] args) {

        // Get the current date

        LocalDate currentDate = LocalDate.now();

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

        // Get the current date and time

        LocalDateTime currentDateTime = LocalDateTime.now();

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

        // Format the current date and time

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

        String formattedDateTime = currentDateTime.format(formatter);

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

        // Parse a date string

        String dateString = "2024-07-21";

        LocalDate parsedDate = LocalDate.parse(dateString, DateTimeFormatter.ISO\_LOCAL\_DATE);

        System.out.println("Parsed Date: " + parsedDate);

    }

}

### Explanation:

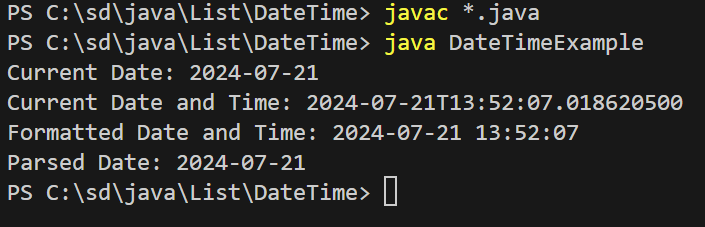
1. **Getting the Current Date and Time**:
   * **LocalDate.now()** returns the current date.
   * **LocalDateTime.now()** returns the current date and time.
2. **Formatting the Date and Time**:
   * **DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")** creates a formatter with the specified pattern.
   * **currentDateTime.format(formatter)** formats the current date and time using the formatter.
3. **Parsing a Date String**:
   * **LocalDate.parse(dateString, DateTimeFormatter.ISO\_LOCAL\_DATE)** parses the date string into a ***LocalDate*** object using the ***ISO local date format***.

### Execute of the Program

You can run this program to see how it works with dates and times in Java.

**javac \*.java**

**java DateTimeExample**

****

# Lambda Expressions and Functional Interfaces

By using **lambda expressions**, you can **easily pass** different behaviors (commands) to **methods** **without having to create new classes** or detailed implementations each time

### Functional Interface Analogy:

Imagine you are a manager at a company, and you need to assign tasks to your employees. You have a set of predefined tasks that need to be done, such as "print a report", "send an email", or "schedule a meeting".

A **functional interface** in this analogy would be like a job description. For example, the job description might say, "**This job involves performing an action with a specified item.**" In technical terms, it would be an interface with a single abstract method, like "**performTask**(String item)".

### Lambda Expression Analogy:

Now, instead of writing detailed instructions for each employee every time you assign a task, you can give a brief, direct instruction that fits the job description. This is where **lambda expressions** come in.

A lambda expression would be like giving a quick, specific command that matches the job description. For example:

* If the task is to "print a report", you might simply say, "Print the report."
* If the task is to "send an email", you might say, "Send an email to the client."

These commands are quick and to the point, just like lambda expressions in programming. They fulfill the requirement of the job description (functional interface) without needing a lengthy explanation.

****

### Putting It Together:

1. **Functional Interface (Job Description)**:
   * Defines the general kind of task that needs to be done.
   * Example: "Perform an action with an item."
2. **Lambda Expression (Quick Command)**:
   * Provides a specific instruction that fits the job description.
   * Example: "Print the report" or "Send an email to the client."

### Putting It Together:

**LambdaExample.java**

// LambdaExample.java

@FunctionalInterface

interface Greeting {

    void sayHello(String name);

}

public class LambdaExample {

    public static void main(String[] args) {

        // Using a lambda expression to implement the sayHello method

        Greeting greeting = (name) -> System.out.println("Hello, " + name);

        // Call the method

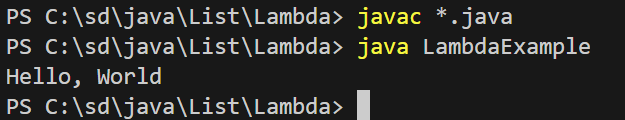
        greeting.sayHello("World");

    }

}

**javac \*.java**

**java LambdaExample**

****

### Explanation:

1. **Functional Interface**:
   * The Greeting interface is defined with the @FunctionalInterface annotation and a single abstract method sayHello(String name).
2. **Lambda Expression**:
   * Inside the main method of the **LambdaExample** class, a lambda expression is used to implement the **sayHello method** of the Greeting interface.
3. **Usage**:
   * The **lambda expression** **(name) -> System.out.println("Hello, " + name)** is assigned to a Greeting reference.
   * The **sayHello** method is called with the **argumen**t "World", which prints "Hello, World" to the console.