Format a given Date and Time

Estimated time needed: 40 minutes

After completing this lab, you will know how to work with dates and times in different formats and manipulate the dates.

You are currently viewing this lab in a Cloud-based Integrated Development Environment (Cloud IDE). It is a fully-online integrated development environment that is pre-installed with JDK 21, allowing you to code, develop, and learn in one location.

Learning Objectives

After completing this lab, you will be able to:

- Retrieves the current date and time using Java's date/time classes.
- Display the dates in standardized formats (for example, yyyy-MM-dd).
- Input dates in various formats (for example, MM/dd/yyyy).
- Add or subtract days from the current date and display the resulting date.
- Convert between time zones.

LocalDate and LocalTime Classes

Date and time in Java is handled with the LocalDate class and LocalTime class in `java.time` package, respectively. In this section of the lab you will write a program that retrieves the current date and time using Java's date/time classes.

1. Create a project directory by running the following command.

   ```
   mkdir my_datetime_proj
   ```

2. Run the following code to create the directory structure.

   ```
   mkdir -p my_datetime_proj/src
   mkdir -p my_datetime_proj/classes
   mkdir -p my_datetime_proj/test
   cd my_datetime_proj
   ```

3. Now, create a file named `DateTimePrinter.java` inside the src directory.

   ```
   touch /home/project/my_datetime_proj/src/DateTimePrinter.java
   ```

4. Click the following button called **Open DateTimePrinter.java in IDE** to open the file for editing.

Open **DateTimePrinter.java** in IDE

5. Read each statement and the accompanying explanations in the following program to understand how to handle LocalDate and LocalTime. Paste the following content in `DateTimePrinter.java`.

   ```
   // Import the LocalDate class from the java.time package to work with dates
   import java.time.LocalDate;
   // Import the LocalTime class from the java.time package to work with times
   import java.time.LocalTime;
   public class DateTimePrinter {
       public static void main(String s[]) {
   ```

```
        // Print the current date to the console
        // LocalDate.now() retrieves the current date from the system clock
        System.out.println("The date is " + LocalDate.now());
        // Print the current time to the console
        // LocalTime.now() retrieves the current time from the system clock
        System.out.println("The time is " + LocalTime.now());
    }
}
```

`LocalDate.now()` and `LocalTime.now()` are static methods that fetch the current date and time from the system clock.

This program is a simple demonstration of how to retrieve and display the current date and time in Java.

6. In the IDE, compile the Java program. specifying the destination directory as the `classes` directory that you created. See the following code example.

   ```
   javac -d classes src/DateTimePrinter.java
   ```

7. Set the `CLASSPATH` variable.

   ```
   export CLASSPATH=$CLASSPATH:/home/project/my_datetime_proj/classes
   ```

8. Now, when you run the Java program, the Java program will run seamlessly as expected.

   ```
   java DateTimePrinter
   ```

You will see the following output:

```
The date is 2025-02-09
The time is 16:54:38.778856
```

Format the date and time

You can choose the format in which the date and time are printed, other than the default format which is how it was printed out in the previous section.

1. Click the following button called **Open DateTimePrinter.java in IDE** to open the file for editing if the IDE is not already open.

Open **DateTimePrinter.java** in IDE

2. Read each statement in the following program to understand how to change the format in which the date and time are returned. Paste the following content in `DateTimePrinter.java`.

   ```
   // Import the LocalDate class from the java.time package to work with dates
   import java.time.LocalDate;
   ```

```
// Import the LocalTime class from the java.time package to work with times
import java.time.LocalTime;
// Import the LocalDateTime class from the java.time package to work with both date and time
import java.time.LocalDateTime;
// Import the DateTimeFormatter class to format date and time
import java.time.format.DateTimeFormatter;
// Import the Scanner class to read user input
import java.util.Scanner;
// Define a class named DateTimePrinter
public class DateTimePrinter {
    // Main method, the entry point of the program
    public static void main(String s[]) {
        // Get the current date using LocalDate.now()
        LocalDate todayDate = LocalDate.now();
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);
        // Prompt the user to enter the preferred date format
        System.out.println("Enter the format you would like to print the date in\n" +
                "dd for date, \n" +
                "M for month, MM for zero-padded month, MMM for abbreviated month, MMMM for full name, \n" +
                "yy or yyyy for year");
        // Read the date format input from the user
        String strDateFormat = scanner.nextLine();
        // Create a DateTimeFormatter object using the user-provided format
        DateTimeFormatter newdateformat = DateTimeFormatter.ofPattern(strDateFormat);
        // Format and print the current date using the specified format
        System.out.println("\nThe date is " + todayDate.format(newdateformat));
        // Get the current time using LocalTime.now()
        LocalTime nowTime = LocalTime.now();
        // Prompt the user to enter the preferred time format
        System.out.println("\n\nEnter the format you would like to print the time in\n" +
                "H for Hour of day (0-23), HH for Zero-padded hour of day (00-23), \n" +
                "h for Hour of am/pm (1-12), hh for Zero-padded hour of am/pm (01-12) \n" +
                "m for Minute of hour (0-59)\n" +
                "mm for Zero-padded minute of hour (00-59)\n" +
                "s for Second of minute (0-59), ss for Zero-padded second of minute (00-59)");
        // Read the time format input from the user
        String strTimeFormat = scanner.nextLine();
        // Create a DateTimeFormatter object using the user-provided format
        DateTimeFormatter newtimeformat = DateTimeFormatter.ofPattern(strTimeFormat);
        // Format and print the current time using the specified format
        System.out.println("The time is " + nowTime.format(newtimeformat));
        // Get the current date and time using LocalDateTime.now()
        LocalDateTime nowDateTime = LocalDateTime.now();
        // Print the current date and time in the default format
        System.out.println(nowDateTime);
        // Prompt the user to enter the preferred date and time format
        System.out.println("\n\nEnter the format you would like to print the date and time in\n");
        // Read the date and time format input from the user
        String strDateTimeFormat = scanner.nextLine();
        // Create a DateTimeFormatter object using the user-provided format
        DateTimeFormatter newDateTimeformat = DateTimeFormatter.ofPattern(strDateTimeFormat);
        // Format and print the current date and time using the specified format
        System.out.println("The date and time is " + nowDateTime.format(newDateTimeformat));
    }
}
```

Heres an explanation of the code:

You will notice that in addition to `LocalDate` and `LocalTime`, you also have `LocalDateTime` which has both the current date and time informention.

The program prompts the user to enter custom formats for Date. The user can specify how the date should be displayed (for example, dd/MM/yyyy, MMM dd, yyyy).

The program prompts the user to enter custom formats for Time. The user can specify how the time should be displayed (for example, HH:mm:ss, hh:mm a).

The program prompts the user to enter custom formats for Date and time together. The user can specify how both date and time should be displayed (for example, dd MMM yyyy HH:mm:ss).

`DateTimeFormatter.ofPattern(String pattern)` is used to create a formatter based on the user-provided pattern.

`format()` method is used to apply the formatter to the current date, time, or date-time.

3. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/DateTimePrinter.java
```

4. Now, run the Java program.

```
java DateTimePrinter
```

Please see the following sample output.

```
Enter the format you would like to print the date in
dd for date,
M for month, MM for zero-padded month, MMM for abreviated month, MMMM for full name,
yy or yyyy for year
dd MM yy
The date is 09 02 25
Enter the format you would like to print the time in
H for Hour of day (0-23), HH for Zero-padded hour of day (00-23),
h for Hour of am/pm (1-12), hh for Zero-padded hour of am/pm (01-12)
m for Minute of hour (0-59)
mm for Zero-padded minute of hour (00-59)
s for Second of minute (0-59), ss for Zero-padded second of minute (00-59)
hh:mm:ss
The time is 05:43:31
2025-02-09T17:43:39.618124
Enter the format you would like to print the time in
yyyy, dd MMM hh:mm
The date and time is 2025, 09 Feb 05:43
```

You can try running with various permuations and combinations.

ZonedDateTime class

When you are working across time zone, you can use `ZonedDateTime` which is also available in `java.time`.

1. Create a file named `ZonedDateTimePrinter.java` inside the src directory.

```
touch /home/project/my_datetime_proj/src/ZonedDateTimePrinter.java
```

2. Click the following button called **Open ZonedDateTimePrinter.java in IDE** to open the file for editing if the IDE is not already open.

Open **ZonedDateTimePrinter.java** in IDE

3. Read each statement in the following program to understand the additional options that you can pass for ZonedDateTime formatting. Paste the following content in `ZonedDateTimePrinter.java`.

```
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;
public class ZonedDateTimePrinter {
    public static void main(String s[]) {
        Scanner scanner = new Scanner(System.in);
        ZonedDateTime nowDateTime = ZonedDateTime.now();
        System.out.println("\nDefault Format " + nowDateTime);
        System.out.println("\n\nEnter the format you would like to print the time in\n" +
                           "dd for date, \n"+
                           "M for month, MM for zero-padded month, MMM for abreviated month, MMMM for full name, \n" +
                           "yy or yyyy for year\n" +
                           "H for Hour of day (0-23), HH for Zero-padded hour of day (00-23), \n" +
                           "h for Hour of am/pm (1-12), hh for Zero-padded hour of am/pm (01-12) \n" +
                           "m for Minute of hour (0-59)\n" +
                           "mm for Zero-padded minute of hour (00-59)\n" +
                           "s for Second of minute (0-59), ss for Zero-padded second of minute (00-59)\n" +
                           "z for Time zone (for example, PDT, EST), zzz: Time zone (for example, Pacific Daylight Time, Eastern Standard Time)" +
                           "Z for Time zone offset (for example, +0800, -0500)");
        String strDateTimeFormat = scanner.nextLine();
        DateTimeFormatter newDateTimeformat = DateTimeFormatter.ofPattern(strDateTimeFormat);
        System.out.println("The date and time is " + nowDateTime.format(newDateTimeformat));
    }
}
```

This program allows the user to input a custom date-time format and then prints the current date and time in that format. The program allows the user to specify any valid date-time format, making it highly flexible. Since ZonedDateTime is used, the output includes time zone information.

4. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/ZonedDateTimePrinter.java
```

5. Now, run the Java program.

```
java ZonedDateTimePrinter
```

Please refer to the following sample output.

```
Default Format 2025-02-09T18:34:03.957761+11:00[Australia/Melbourne]
Enter the format you would like to print the time in
dd for date,
M for month, MM for zero-padded month, MMM for abreviated month, MMMM for full name,
yy or yyyy for year
H for Hour of day (0-23), HH for Zero-padded hour of day (00-23),
h for Hour of am/pm (1-12), hh for Zero-padded hour of am/pm (01-12)
m for Minute of hour (0-59)
mm for Zero-padded minute of hour (00-59)
s for Second of minute (0-59), ss for Zero-padded second of minute (00-59)
z for Time zone (for example, PDT, EST), zzz: Time zone (for example, Pacific Daylight Time, Eastern Standard Time)Z for Time zone offset (for example
dd MMM yy hh:mm:ss zzz
The date and time is 09 Feb 25 06:34:03 AEDT
```

Add or subtract date and time

The date and time are nano from Unix EPOCH time in Java (January 1, 1970, 00:00:00 UTC.)

1. Create a file named `DateTimeOperations.java` inside the src directory.

```
touch /home/project/my_datetime_proj/src/DateTimeOperations.java
```

2. Click the following button called **Open DateTimeOperations.java in IDE** to open the file for editing if the IDE is not already open.

Open **DateTimeOperations.java** in IDE

3. Read each statement in the following program to understand how to add and delete units of time. Paste the following content in `DateTimeOperations.java`.

```
// Import necessary classes for date-time manipulation and user input
import java.time.ZonedDateTime;
import java.util.Scanner;
import java.time.format.DateTimeFormatter;
import java.time.ZoneId;
public class DateTimeOperations {
    public static void main(String s[]) {
        Scanner scanner = new Scanner(System.in);
```

```
// Get the current date and time in the default time zone
ZonedDateTime dateTime = ZonedDateTime.now();
// Print the current date and time in the default format
System.out.println("\nDefault Format: " + dateTime);
// Loop indefinitely until the user chooses to exit
while (true) {
    // Display the menu options to the user
    System.out.println("\nPress 1 for adding to or deleting from the current date/time," +
            "\nPress 2 for changing the timezone," +
            "\nAny other key to exit");
    // Read the user's choice
    String userAction = scanner.nextLine();
    // Option 1: Add or delete time units
    if (userAction.equals("1")) {
        // Prompt the user to select a unit of time to modify
        System.out.println("Enter which unit of time you want to add/delete:\n" +
                "d - days, M - months, y - years, h - hours, m - minutes, s - seconds, w - weeks");
        String unitToChange = scanner.nextLine();
        // Prompt the user to enter the quantity of the selected unit
        System.out.println("Enter the quantity to change (for example, 3):");
        long qtyToChange = Long.parseLong(scanner.nextLine());
        // Prompt the user to choose whether to add or delete the selected quantity
        System.out.println("\nPress 'a' to add, 'd' to delete," +
                "\nAny other key to go back to the main menu");
        String addOrDel = scanner.nextLine();
        // Variable to store the modified date and time
        ZonedDateTime newDateTime = null;
        // Handle the user's choice to add or delete time
        if (addOrDel.equals("a")) {
            // Add the specified quantity of the selected unit
            switch (unitToChange) {
                case "d":
                    newDateTime = dateTime.plusDays(qtyToChange);
                    break;
                case "M":
                    newDateTime = dateTime.plusMonths(qtyToChange);
                    break;
                case "y":
                    newDateTime = dateTime.plusYears(qtyToChange);
                    break;
                case "h":
                    newDateTime = dateTime.plusHours(qtyToChange);
                    break;
                case "m":
                    newDateTime = dateTime.plusMinutes(qtyToChange);
                    break;
                case "s":
                    newDateTime = dateTime.plusSeconds(qtyToChange);
                    break;
                case "w":
                    newDateTime = dateTime.plusWeeks(qtyToChange);
                    break;
                default:
                    System.out.println("Invalid input");
                    continue; // Go back to the main menu
            }
        } else if (addOrDel.equals("d")) {
            // Subtract the specified quantity of the selected unit
            switch (unitToChange) {
                case "d":
                    newDateTime = dateTime.minusDays(qtyToChange);
                    break;
                case "M":
                    newDateTime = dateTime.minusMonths(qtyToChange);
                    break;
                case "y":
                    newDateTime = dateTime.minusYears(qtyToChange);
                    break;
                case "h":
                    newDateTime = dateTime.minusHours(qtyToChange);
                    break;
                case "m":
                    newDateTime = dateTime.minusMinutes(qtyToChange);
                    break;
                case "s":
                    newDateTime = dateTime.minusSeconds(qtyToChange);
                    break;
                case "w":
                    newDateTime = dateTime.minusWeeks(qtyToChange);
                    break;
                default:
                    System.out.println("Invalid input");
                    continue; // Go back to the main menu
            }
        } else {
            // Handle invalid input
            System.out.println("Invalid input");
            continue; // Go back to the main menu
        }
        // Prompt the user to enter a custom format for the modified date and time
        System.out.println("\n\nEnter the format you would like to print the date and time in:");
        String strDateTimeFormat = scanner.nextLine();
        // Create a DateTimeFormatter object using the user-provided format
        DateTimeFormatter newDateTimeformat = DateTimeFormatter.ofPattern(strDateTimeFormat);
        // Print the modified date and time in the specified format
        System.out.println("The date and time is: " + newDateTime.format(newDateTimeformat));
    }
    // Option 2: Change the time zone
    else if (userAction.equals("2")) {
        // Display all available time zone IDs
        System.out.println("Check the following Zone options:");
        Object[] zoneIds = ZoneId.getAvailableZoneIds().toArray();
        for (Object zoneId : zoneIds) {
            System.out.println(zoneId);
        }
        // Prompt the user to enter a new time zone
        System.out.println("\n\nEnter the time zone you want to get the current time for:");
        String newZone = scanner.nextLine();
        // Convert the current date and time to the new time zone
```

```
                ZonedDateTime newZoneDateTime = dateTime.withZoneSameInstant(ZoneId.of(newZone));
                // Define a default format for displaying the date and time
                DateTimeFormatter newDateTimeformat = DateTimeFormatter.ofPattern("dd MMM yyyy hh:mm:ss a zzz");
                // Print the date and time in the new time zone using the default format
                System.out.println("The date and time is: " + newZoneDateTime.format(newDateTimeformat));
            }
            // Exit the program
            else {
                break;
            }
        }
    }
}
```

Operations

LocalDateTime and ZoneDateTime provides with many methods. In this program, you will use the following to add or delete units of time.

- `plusDays`, `minusDays`
- `plusWeeks`, `minusWeeks`
- `plusMonths`, `minusMonths`
- `plusYears`, `minusYears`
- `plusHours`, `minusHours`
- `plusMinutes`, `minusMinutes`
- `plusSeconds`, `minusSeconds`

switch-case

The switch-case statement provides a concise and readable way to handle multiple cases, rather than using mutiple if-else blocks, making the code more efficient and easier to maintain.

- The `switch` keyword is followed by an expression, which is `unitToChange` in this case. This expression is evaluated to determine which case to run.
- The `case` keyword is followed by a label, which is a value that the switch expression can take. In this code, the case labels are "d", "M", "y", etc.
- When the switch expression matches a case label, the code associated with that case is run. In this example, the code adds or subtracts a certain amount of time from the dateTime object.
- The break statement is used to exit the switch block and continue executing the code after the switch statement. If the break statement is omitted, the code will continue executing the next case, which can lead to unexpected behavior.
- Default case: The default case is optional and is run when the switch expression does not match any of the case labels. In this example, the default case prints an error message and continues to the next iteration of the loop.

Handling multiple zones

You will use the ZoneId class to get all the valid zones. You will notice how you are provided with a `for` loop without initial value, condition and increment. It is called the `for-each` loop. The `for-each` loop iterates over each element in the collection. This is very useful for iterative collections.

```
Object[] zoneIds = ZoneId.getAvailableZoneIds().toArray();
for (Object zoneId : zoneIds) {
    System.out.println(zoneId);
}
```

The `for-each` loop iterates over each element in the zoneIds array. Since the array contains Object's, each element is assigned to the variable `zoneId` of type Object.

4. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/DateTimeOperations.java
```

5. Now, run the Java program.

```
java DateTimeOperations
```

Please see the following sample output.

Option 1 add to/delete from current date and time

```
Default Format: 2025-02-09T20:27:42.905214+11:00[Australia/Melbourne]
Press 1 for adding to or deleting from the current date/time,
Press 2 for changing the timezone,
Any other key to exit
1
Enter which unit of time you want to add/delete:
d - days, M - months, y - years, h - hours, m - minutes, s - seconds, w - weeks
y
Enter the quantity to change (for example, 3):
4
Press 'a' to add, 'd' to delete,
Any other key to go back to the main menu
d
Enter the format you would like to print the date and time in:
dd MMM yyyy hh:mm:ss a
The date and time is: 09 Feb 2021 08:27:42 pm
Press 1 for adding to or deleting from the current date/time,
Press 2 for changing the timezone,
Any other key to exit
```

Option 2 Change TimeZone

```
Default Format: 2025-02-09T20:28:44.438751+11:00[Australia/Melbourne]
Press 1 for adding to or deleting from the current date/time,
Press 2 for changing the timezone,
Any other key to exit
2
Check the following Zone options:
Asia/Aden
America/Cuiaba
Etc/GMT+9
Etc/GMT+8
.
.
.
.
US/Pacific
Europe/Monaco
Enter the time zone you want to get the current time for:
America/Toronto
The date and time is: 09 Feb 2025 04:28:44 am GMT-05:00
Press 1 for adding to or deleting from the current date/time,
Press 2 for changing the timezone,
Any other key to exit
```

Practice Exercise

Exercise 1

1. Add code to `ZonedDataTimePrinter` class to include `k` options and `a` option in the time formatting.

▶ Click here for solution

Conclusion

In this lab, you learned:

- How to retrieve the current date and time using Java's date/time classes.
- How to display the dates in standardized formats (for example, yyyy-MM-dd).
- How to print dates in various user-chosen formats (for example, dd-MMM-yyyy).
- How to add units of time to or subtract units of time from the current date and display the resulting date.
- How to convert between time zones.

Author(s)

Lavanya